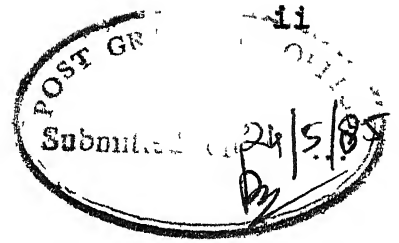# EXACT AND HEURISTIC ALGORITHMS FOR THE OPTIMUM COMMUNICATION SPANNING TREE PROBLEM

A Thesis Submitted

In Partial Fulfilment of the Requirements

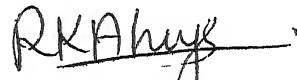for the Degree of

MASTER OF TECHNOLOGY

by

V V. S N MURTY

to the

INDUSTRIAL AND MANAGEMENT ENGINEERING PROGRAMME

INDIAN INSTITUTE OF TECHNOLOGY, KANPUR

MAY, 1985

CERTIFICATE

This is to certify that the work embodied in the thesis entitled, EXACT AND HEURISTIC ALGORITHMS FOR THE OPTIMUM COMMUNICATION SPANNING TREE PROBLEM, by V.V.S.N. Murty has been carried out under my supervision and has not been submitted elsewhere for the award of a degree.

(R. K. Ahuja)
Industrial and Management Engg.Program
Indian Institute of Technology,
Kanpur 208016, INDIA

May, 1985

IMEP-1985-M-MUR-EXA

ACKNOWLEDGEMENTS

CONTENTS

ABSTRACT

The network design problems have been widely investigated in the mathematical programming literature in the last decade. In this thesis, we study one such design problem known as Optimum Communication Spanning Tree (OCST) problem. We develop an exact algorithm based on branch and bound approach and heuristic algorithms for solving the OCST problem. We first derive two new lower planes for the network design problem through combinatorial arguments. One of the lower planes is used in computing lower bounds of subproblems in the branch and bound algorithm. Reoptimization of subproblems is extensively used to reduce computations. The heuristic algorithm is a two-phase algorithm: tree-building algorithm and tree-improvement algorithm. The complexity of these algorithms is analysed. We also investigate a heuristic algorithm which is a branch and bound algorithm with inexact lower bounds.

All these algorithms are programmed in FORTRAN-IV and implemented on DEC-10 computer system. The algorithms are tested on randomly generated Euclidean and non-Euclidean network problems. The branch and bound algorithm is able to solve 40 node, 69 arc Euclidean network problem in about 37 minutes of execution time. Non-Euclidean problems are solved more easily. The two-phase heuristic algorithm has produced excellent results. All the problems tested are solved

optimally. The computational times are also very reasonable. Problems of size 100 nodes and 1,000 arcs are solved in about 30 seconds of execution time. The performance of the heuristic algorithm based on inexact bounds is not encouraging.

CHAPTER I

INTRODUCTION

1.1 INTRODUCTION:

In recent years, much interest has been focussed on various problems that arise in the area of network design. In its simplest form, the network design problem may be stated as the building of a connected sub-network, by selection from the set of arcs of a network, that minimizes the weighted sum of shortest path distances between all pairs of nodes of the sub-network, subjected to a budget constraint which limits the number of arcs that may be included in the sub-network. Such problems find application in transportation planning, communication system planning, water resource planning and distribution planning.

In this dissertation, we consider one such network design problem known as Optimum Communication Spanning Tree (OCST) problem. Given an undirected network and the amount of communication between each pair of nodes, the OCST problem is to determine a spanning tree for which the total cost of communication between all pairs of nodes is minimum among all spanning trees. Such problems are encountered very frequently in designing road, communication and computer networks. In

communication networks, the nodes represent cities and the
amount of communication may be taken as the number of telephone
calls between pairs of cities. The total cost of communication
decreases if more number of arcs are included in the sub-network
but the construction cost limits the inclusion of more number
of arcs. Thus a sub-network is to be built which barely connects
all the cities and at the same time the total communication cost
is minimum. This results in building a spanning tree that
minimizes the total cost of communication among all spanning
trees.

In all these applications, the actual design problem is
usually subjected to many constraints but the solution of the
OCST problem may be used as a measuring standard for the effi-
ciency of proposed designs. The OCST problem has been shown to
be NP-complete [14] even for the special case when the amount
of communication is same for all pairs of nodes.

In this dissertation, we develop exact and heuristic
algorithms for the OCST problem. The exact algorithm is based
on the branch and bound approach for solving combinatorial
optimization problems. A new lower bounding strategy is
developed for the general network design problem which is used
in the branch and bound algorithm. The heuristic algorithm is
a two-phase algorithm - the tree-building algorithm and the
tree-improvement algorithm. The tree-building algorithm starts
with a seed node and successively builds the tree by adding one

arc each time such that a good feasible solution is obtained. The spanning tree thus constructed is fed to the tree improvement algorithm which examines each tree arc one by one and tries to interchange it with an appropriate non-tree arc so that the cost of communication decreases. The algorithm terminates when no possible interchange yields improvement. Another heuristic algorithm, named Global heuristic, is also developed. It is essentially the branch and bound algorithm with the modification that in-exact lower bound is used. This is an adaptation of the heuristic proposed by Dionne and Florian [5] for the network design problem and was claimed to be giving excellent results.

All the suggested algorithms have been programmed and tested extensively on randomly generated network problems. The branch and bound algorithm is able to solve 40 nodes, 69 arcs problem. For all the problems solved by the branch and bound algorithm, the two-phase heuristic algorithm gave the optimal solution. The global heuristic algorithm was observed to be taking more time when compared to the two-phase heuristic algorithm. Detailed computational results are presented.

1.2 PRELIMINARIES:

Some notations and well-known concepts of graph theory, used throughout the thesis, are given below.

A graph G = (N, A) consists of a finite set N of

elements, called nodes, and a set A of pairs of nodes called arcs. An undirected graph is a graph in which the arcs have no direction.

A graph in which numbers are associated with nodes and arcs is called a network. A path in G = (N, A) is a sequence $i_1, i_2, \ldots, i_r$ of distinct nodes of N such that $(i_k, i_{k+1}) \in A$, for each k = 1, ..., r-1. A cycle is a path together with an arc (i_r, i_1).

A graph G' = (N', A') is a subgraph of G = (N, A) if $N' \subseteq N$ and $A' \subseteq A$. A graph is connected if there is a path between any two nodes of the graph. A tree of k nodes is a graph which is connected and has k - 1 arcs. A sub-graph which is a tree and contains all vertices of a graph is called a spanning tree. (In a spanning tree, there is a unique path between any two nodes). Let,

$m = |A|, \quad n = |N|$

r_{ij} = amount of communication between the pair of nodes i and j,

d_{ij} = length of the arc [i,j], $i \neq j$ (= ∞ if there is no arc between i and j)

d_{ii} = 0, for i = 1, ..., n,

$\tilde{A}$ = a subset of A

$u_{ij}(\tilde{A})$ = shortest distance between the pair of nodes i and j over the graph G(N, $\tilde{A}$).

The following algorithms are extensively used in the present work and are given below.

1. Floyd's [7] Algorithm for Finding Shortest Path Distances between all pairs of nodes:

Procedure FLOYD (n, d, u);

begin

Set $u_{ij} := d_{ij}$, $\forall$ i := 1,..., n, and $\forall$ j := 1,..., n.

for h := 1 to n do

for i := 1 to n-1 do

for j := i+1 to n do

if $u_{ij} > u_{ih} + u_{hj}$ then

begin

$u_{ij} := u_{ih} + u_{hj}$; $u_{ji} := u_{ij}$

end

end;

2. Dionne and Florian's [5] Algorithm for Updating the Shortest path distances when an arc length changes:

Let [p,q] be the modified arc, its length being d_{pq} before modification and d'_{pq} afterwards. The algorithm obtains the modified u matrix when the length of the arc changes from d_{pq} to d'_{pq} procedure ADDARC for the case when $d_{pq} > d'_{pq}$ and procedure DELETEARC for the case when $d_{pq} < d'_{pq}$.

```
Procedure ADDARC;

begin

    N1:= ∅;  N2:= ∅;

    for i:= 1 to n do

        if u_iq > u_ip + d'_pq   then

            begin

                N1:= N1 ∪ {i};  u_iq:= u_ip + d'_pq;  u_qi := u_iq

            end;

    for j:= 1 to n do

        if u_pj > d'_pq + u_qj  then

            begin

                N2:= N2 ∪ {j};  u_pj:= d'_pq + u_qj;  u_jp:= u_pj

            end;

    for every (i,j) ε N1 x N2 do

        if u_ij > u_ip + u_pj  then

            begin

                u_ij:= u_ip + u_pj;  u_ji:= u_ij

            end

end;

Procedure DELETEARC;

begin

    N1:= ∅;  N2:= ∅;  R:= ∅;

    for i:= 1 to n do

        if u_iq = u_ip + d_pq then N1:= N1 ∪ {i};

    for j:= 1 to n do

        if u_pj = d_pq + u_qj then N2:= N2 ∪{j};
```

```
for every (i,j) ε N1 x N2 do

   if u_ij = u_ip + d_pq + u_qj then
      begin
         R:= R U {(i,j)}; u_ij:= d_ij; u_ji:= u_ij;
      end;
   d_pq:= d'_pq; d_qp:= d_pq; u_pq:= d_pq; u_qp:= u_pq;
   for h:= 1 to n do

      for every (i,j) ε R do

         if u_ih + u_hj < u_ij then
            begin
               u_ij:= u_ih + u_hj; u_ji:= u_ij
            end

end;
```

This procedure DELETEARC is modified to delete a number of arcs at a time and its given as procedure DELETEARCS.

```
Procedure DELETEARCS (P,q);
{Modifies the shortest distance matrix u when all arcs
[p,q], p ε P are removed from the network, R is the
set of retained pairs}
begin
   R:= ∅;
   for every p ε P do
      begin
         N1:= ∅; N2:= ∅;
         for i:= 1 to n do if u_iq = u_ip + d_pq then
            N1:= N1 U {i};
```

```
for j:= 1 to n do if u_pj = d_pq + u_qj then
    N2:= N2 ∪ {j};
for every (i,j) ε N1 x N2 do
    begin
        if u_ip + d_pq + u_qj = u_ij then R:= R ∪ {(i,j)}
    end;
    d_pq:= d'_pq; d_qp:= d_pq
end;
```

$$\text{for } j:= 1 \text{ to } n \text{ do if } u_{pj} = d_{pq} + u_{qj} \text{ then}$$
$$N2:= N2 \cup \{j\};$$
$$\text{for every } (i,j) \in N1 \times N2 \text{ do}$$
$$\text{begin}$$
$$\text{if } u_{ip} + d_{pq} + u_{qj} = u_{ij} \text{ then } R:= R \cup \{(i,j)\}$$
$$\text{end;}$$
$$d_{pq}:= d'_{pq}; \quad d_{qp}:= d_{pq}$$
$$\text{end;}$$

$$\text{for every } (i,j) \in R \text{ do}$$
$$\text{begin}$$
$$u_{ij}:= d_{ij}; \quad u_{ji}:= u_{ij}$$
$$\text{end;}$$

$$\text{for } h:= 1 \text{ to } n \text{ do}$$
$$\text{for every } (i,j) \in R \text{ do}$$
$$\text{if } u_{ih} + u_{hj} < u_{ij} \text{ then}$$
$$\text{begin}$$
$$u_{ij}:= u_{ih} + u_{hj}; \quad u_{ji}:= u_{ij}$$
$$\text{end;}$$

$$\text{end;}$$

3. Kruskal's [15] Algorithm for Minimum Spanning Tree Problem:

Procedure MST;

begin

$T:= \emptyset$; $E:= A$;

While $|T| < n-1$ and $E \neq \emptyset$ do

```
begin
    find an edge e in E with minimum weight;
    E:= E-{e};
    if T U {e} has no cycle then T:= T U {e}
end
```

end;

The maximum spanning tree problem is solved using Kruskal's algorithm and the data structures given in [24] are employed for implementing this algorithm.

1.3 STATEMENT OF THE PROBLEM:

Optimum Communication Spanning Tree (OCST) Problem:

Given an undirected network and the amount of communication, r_{ij}, between each pair of nodes i and j, the OCST problem is to determine a spanning tree, for which the total cost of communication between all pairs of nodes is minimum among all spanning trees. The cost of a given spanning tree is defined as follows. For a pair of nodes i and j, there is a unique path in the spanning tree between i and j. The length of the path is the sum of lengths of arcs in the path. The cost of communication for the pair of nodes i and j is r_{ij} multiplied by the length of the path. Summing over all $n(n-1)/2$ pairs of nodes, we get the cost of the spanning tree. This can be stated as,

$$\text{Min. } \frac{1}{2} \sum_{i=1}^{n} \sum_{j=1}^{n} r_{ij} \cdot u_{ij} \ (\tilde{X})$$

$\tilde{X}$ is a spanning tree.

Two special cases of the OCST problem may be stated as:

1. Optimum Requirement Spanning Tree: The arc lengths d_{ij} are all equal to 1, while the requirements r_{ij} are arbitrary.

2. Optimum Distance Spanning Tree: The r_{ij} are all equal to 1, while the d_{ij} are arbitrary.

The general OCST problem has been formulated by Hu [13]. He has also developed an $O(n^4)$ algorithm for solving the optimum requirement spanning tree problem. The optimum distance spanning tree problem has been shown to be NP-complete by Johnson, Lenstra and Rinnooy Kan [14] by showing that the Exact 3-Cover is reducible to the above problem.

1.4 OUTLINE OF THE THESIS:

A brief chapter by chapter outline of the thesis is given in this section.

In Chapter II, existing lower planes for the network design problem are described and two new lower planes are proposed. An $O(mn^2)$ lower plane by Hoang [11] and an $O(n^4)$ lower plane by Gallo [9] have been proposed for the network design problem in the literature. A new lower plane of worst case complexity $O(n^4)$ which is tighter than that of Gallo is developed in Section 2.3. Another lower plane of $O(n^3)$

complexity is proposed in Section 2.4. A comparison of lower bounds by all the methods for randomly generated problems is presented in Section 2.5.

In Chapter III, we develop an exact algorithm based on the branch and bound approach and two heuristic algorithms - (i) Two-Phase heuristic and (ii) Global heuristic. The new lower plane developed in Section 2.3. is used for lower bound calculation. Some of the important features of the branch and bound algorithm are:

(i)   Sensitivity analysis is used to update the shortest path distance matrix whenever an arc length changes (increases or decreases).

(ii)   Reoptimization is done to reduce the computations substantially.

(iii)   The new lower plane of Section 2.3 along with the maximum spanning tree algorithm is used in the lower bound calculation.

(iv)   The arc selected for branching is such that the pruning is more effective.

(v)   Depth first search strategy is employed to reduce the storage requirements.

Detailed discussion of all these aspects is presented. A numerical example is given to clarify the above steps.

In the first phase of the Two-Phase heuristic algorithm, a spanning tree is constructed by adding one arc each time such that a good feasible solution is obtained. In the second phase,

each tree-arc is examined one-by-one and an interchange of the tree-arc with an appropriate non-tree arc is made if the cost of communication decreases by such an interchange. The algorithm terminates when no such interchange yields improvement. The global heuristic algorithm is same as the branch and bound algorithm except that an in-exact lower bounding is employed. Because of the increased lower bound, the algorithm takes less computational time as compared with the branch and bound algorithm, but the solution may not be optimal.

In the same chapter, we present the computational performance of branch and bound, two-phase heuristic and global heuristic algorithms. The results are found to be quite encouraging. The branch and bound algorithm is able to solve 40 nodes and 69 arcs problem in 37 min. of CPU time. For all the problems solved by the branch and bound algorithm, the two-phase heuristic algorithm gave the optimal solution. The global heuristic algorithm was observed to be taking excessive times when compared with the two-phase heuristic algorithm. Computer programs for all these algorithms are written in FORTRAN - IV and implemented on DEC-1090 computer system. Detailed computational results are presented.

CHAPTER II

NEW LOWER PLANES FOR THE NETWORK DESIGN PROBLEM

2.1 INTRODUCTION:

The network design problems have been widely studied in literature [20]. These are difficult combinatorial optimization problems and have been shown to be NP-hard [14]. Benders decomposition [4,6,10,12,18,19,22] and branch and bound approaches [1,2,3,5,8,11,16,17,23] have been the two most effective optimization based solution techniques. However, despite these efforts only moderately sized problems (upto 30 nodes and 60 arcs) have been solved within reasonable computational times. In this chapter, we develop two new lower bounding methods for the network design problem, whose use in the branch and bound algorithms might enable us to solve larger sized problems.

The network design problem, addressed in this chapter, is to identify a subnetwork of a given network G satisfying a budget constraint, over which the cost of transportation is minimum among all such subnetworks. A subproblem in the branch and bound algorithm for the network design problem consists of a set of arcs, I, to be included in the network, another set of arcs, E, to be excluded from the network, and rest of the

arcs, U, unassigned. A naive lower bound μ_0 can be obtained by determining all pair shortest path distances u_{ij} over the network consisting of the arcs in I U U and letting

$$\mu_0 = \frac{1}{2} \sum_{i=1}^{n} \sum_{j=1}^{n} u_{ij} \cdot r_{ij}$$

where r_{ij} is the amount of interaction between the pair of nodes i and j. However, the budget will not permit all unassigned arcs to be included in the network. Therefore, some arcs in U will have to be deleted leading to increase in the shortest path distances and, accordingly, the lower bound value. Hoang [11] has suggested an $O(mn^2)$ method which considers the effect of possible arc deletion on m pairs of nodes. Recently, Gallo [9] has proposed an $O(n^4)$ method which considers the effect of arc deletion on n(n-1)/2 pairs of nodes. The first lower bounding method suggested by us can consider the effect of arc deletion on each pair of nodes more than once and, consequently, obtains a bound which is always sharper than that of Gallo. The complexity of this method is shown to be $O(n^4)$. The second lower bounding method suggested by us is of complexity $O(n^3)$ and produces a bound which is generally worse than that of Gallo, but much sharper than that of Hoang. Computational results are also presented.

2.2 NOTATIONS:

We use the notations similar to those of Gallo [9]. For the sake of completeness, we give them below. Let,

G = (N, A) be an undirected network,

n = $|N|$,

m = $|A|$,

$\tilde{A}$ = a subset of arc set A,

a_k = the k-th arc, also represented as $[i_k, j_k]$,

c_k = cost of constructing the arc a_k,

b = budget for constructing arcs,

d_k = length of the arc a_k,

r_{ij} = amount of interaction between the pair of nodes i and j,

Q = $\{(i,j): 1 \leq i < j \leq n\}$, the set of all pairs of nodes,

$u_{ij}(\tilde{A})$ = shortest distance between the nodes i and j over the graph $\tilde{G} = (N, \tilde{A})$,

$g_{ij}(\tilde{A})$ = the second last node on the shortest path in $G(N, \tilde{A})$ from node i to node j if $i \neq j$, and 0 if $i = j$. These indices can be obtained while computing all pair shortest path distances,

X = a subset of $\tilde{A}$. Define a characteristic vector of X as a m-vector x with components

$$x_k = \begin{cases} 1, & \text{if } a_k \, \varepsilon \, X \\ 0, & \text{otherwise,} \end{cases}$$

$\pi(X, (i,j))$ = increase in the shortest path distance between the nodes i and j when all arcs in X are deleted from the network $G = (N, \tilde{A})$.

$$\mu_o = \sum_{(i,j)\varepsilon Q} r_{ij} \cdot u_{ij}(\tilde{A}).$$

$$L(\tilde{A}, Q) = \sum_{(i,j)\varepsilon Q} r_{ij} \cdot u_{ij}(\tilde{A}).$$

The following results can be easily deduced from the above definitions:

Proposition 1: If $X \subseteq X$, then $\pi(X, (i,j)) \leq \pi(X, (i,j))$,

$$\forall (i,j) \, \varepsilon \, Q.$$

Proposition 2: $L(\tilde{A} - X, Q) = \mu_0 + \sum_{(i,j) \varepsilon Q} r_{ij} \cdot \pi(X, (i,j)).$

The Network Design Problem (NDP) may be defined as to select a set $X \subseteq A$ such that

(i) $L(A - X, Q)$ is minimum, and

(ii) $x \, \varepsilon \, S = \{x: cx \geq \sum_{k=1}^{m} c_k - b,$ and $A - X$ is connected$\}$.

2.3 LOWER PLANES

A subproblem in the branch and bound algorithm for the NDP consists of solving the problem over $G = (N, \tilde{A})$, where $\tilde{A} \subseteq A$. A linear function $h(x) = \mu_0 + \mu \cdot x$ defined over R^m is a lower plane for NDP if $h(x) \leq L(\tilde{A} - X, Q)$ for every $x \, \varepsilon \, S$. Given any two lower planes $h(x)$ and $g(x)$, we say that $h(x)$ dominates $g(x)$ if $h(x) \geq g(x)$ for every $x \, \varepsilon \, S$.

2.3.1 Existing Lower Planes

Hoang's [11] lower plane: $h^1(x) = \mu_0 + \mu^1 x$ is a lower plane for the NDP, where

$$\mu_k^1 = r_{i_k j_k} \cdot \pi(a_k, (i_k, j_k)), \, \forall \, k = 1, \ldots, m.$$

Gallo's [9] lower plane: $h^2(x) = \mu_0 + \mu^2 x$ is a lower plane for the NDP, where

$$\mu_k^2 = \sum_{(i,j) \varepsilon Q_k} \cdot r_{ij} \, \pi(a_k, (i,j)), \, \forall \, k = 1, \ldots, m,$$

and,

$$Q_k = \{(i,j) \in Q : \pi(a_k, (i,j)) > 0 \text{ and } \pi(a_p, (i,j)) = 0,$$
$$\forall\, p = 1, \ldots, k-1\}, \forall\, k = 1, \ldots, m.$$

Clearly, $h^2(x)$ dominates $h^1(x)$ because if $\mu_k^1 > 0$ for some k, then Q_k always includes the node pair (i_k, j_k). Intuitively, $h^1(x)$ considers the effect of arc deletion on m pairs of nodes, whereas $h^2(x)$ considers $n(n-1)/2$ such interactions. Accordingly, Gallo's numerical investigations have indicated that $h^2(x)$ is much sharper than $h^1(x)$.

2.3.2 The Lower Plane LP1:

We derive another lower plane for the NDP which considers the effect of arc deletion on each pair of nodes more than once but, in effect, it considers $n(n-1)/2$ interactions with highest magnitude. For each $(i,j) \in Q$, define

$$\beta_{ij}^o = 0,$$

$$\beta_{ij}^k = \max_{1 \le p \le k} \{\pi(a_p, (i,j))\}, \forall\, k = 1, \ldots, m,$$

and,

$$w_{ij}^k = \max. \{0, \pi(a_k, (i,j)) - \beta_{ij}^{k-1}\}, \forall\, k = 1, \ldots, m.$$

The following result easily follows from the above definitions.

Proposition 3: For any arc a_k, if $w_{ij}^k > 0$, then $\sum_{p=1}^{k} w_{ij}^p = \pi(a_k, (i,j$

Theorem 1: $h^3(x) = \mu_o + \mu^3 x$ is a lower plane for the NDP, where

$$\mu_k^3 = \sum_{(i,j) \in Q} r_{ij} w_{ij}^k, \forall\, k = 1, \ldots, m.$$

Proof: Consider any X such that its characteristic vector $x \varepsilon S$. Further, consider any node pair (i,j). Let s be the highest index such that $x_s = 1$ and $w_{ij}^s > 0$. If no index satisfies these conditions, then the weight contributed by this node pair to deleted arcs is zero. Clearly

$$\sum_{k=1}^{m} w_{ij}^k \cdot x_k = \sum_{k=1}^{s} w_{ij}^k \cdot x_k \leq \sum_{k=1}^{s} w_{ij}^k.$$

Using Proposition 3 and Proposition 1 we get

$$\sum_{k=1}^{m} w_{ij}^k x_k \leq \sum_{k=1}^{s} w_{ij}^k = \pi(a_s, (i,j)) \leq \pi(X, (i,j)).$$

Multiplying this by r_{ij} and adding for all $(i,j) \varepsilon Q$, we obtain,

$$\sum_{(i,j)\varepsilon Q} \cdot r_{ij} \left(\sum_{k=1}^{m} w_{ij}^k x_k\right) \leq \sum_{(i,j)\varepsilon Q} r_{ij} \cdot \pi(X, (i,j)),$$

which after rearranging terms and using Proposition 2 becomes

$$\sum_{k=1}^{m} \left(\sum_{(i,j)\varepsilon Q} r_{ij} w_{ij}^k\right) x_k \leq L(\tilde{A} - X, Q) - \mu_o,$$

or

$$\mu_o + \mu^3 x \leq L(\tilde{A} - X, Q).$$

Theorem 2: The lower plane $h^3(x)$ dominates the lower plane $h^2(x)$.

Proof: Every element $(i,j) \varepsilon Q_k$ has $\pi(a_k, (i,j)) > 0$ and $\pi(a_p, (i,j)) = 0$, $\forall p = 1,\ldots, k-1$. Hence $\beta_{ij}^{k-1} = 0$ and $w_{ij}^k = \pi(a_k, (i,j))$. It immediately follows that $\mu_k^3 \geq \mu_k^2$, $\forall k = 1,\ldots, m$. Further, there may be some $(i,j) \varepsilon Q$ such that $\beta_{ij}^{k-1} > 0$ and $\pi(a_k, (i,j)) > \beta_{ij}^{k-1}$, which will make the inequality $\mu_k^3 \geq \mu_k^2$ strict.

A procedure, named LP1, to determine $h^3(x)$ is stated below. The procedure deletes each arc $a_k \in \tilde{A}$ one by one, retains the pairs of nodes which might be affected by deleting the arcs in a set S, applies Floyd's algorithm [7] for each retained pair to determine the length of the changed shortest path between the pair, and accordingly determines the weight μ_k^3 to be associated with the arc a_k.

Proposition 4: The computational complexity of determining $h^3(x)$ is $O(n^4)$.

Proof: The shortest path between any pair of nodes consists of atmost n - 1 arcs and deletion of only these arcs can change the shortest path between the pair. Hence each pair of nodes will be retained atmost n - 1 times while deleting arcs one by one. Since there are $n(n-1)/2$ pairs of nodes, the total number of retained pairs will be atmost $n(n-1)^2/2$.

The all pair shortest path lengths can be obtained in $O(n^3)$ operations. The operations in loop-1 are of $O(n^2)$ and since it is executed atmost m times, its complexity is $O(mn^2)$. The complexity of loop-2 is $O(n^4)$, as $O(n)$ computations are performed for every retained pair which are atmost $n(n-1)^2/2$ in number. The complexity of loop-3 is clearly $O(n^3)$. Hence the overall complexity of the procedure LP1 is $O(n^4)$.

```
procedure LP1;
begin
    compute all pair shortest path lengths $u_{ij}$'s and the
    corresponding $g_{ij}$'s;
    Set $\beta_{ij} := 0$, $\forall$ i$:= 1,\ldots,$ n, $\forall$ j$:= 1,\ldots,$ n;
    compute $\mu_0 := \frac{1}{2} \sum\limits_{i=1}^{n} \sum\limits_{j=1}^{n} r_{ij} \cdot u_{ij}$;
    for each $a_k \in \tilde{A}$ do
        begin
            p$:= i_k$; q$:= j_k$;
            if $u_{pq} = d_{pq}$ then
                begin
                    temp$:= d_{pq}$; $d_{pq} := \infty$; $d_{qp} := d_{pq}$;
                    N1$:= \phi$; N2$:= \phi$; S$:= \phi$;
                    for i$:= 1$ to n do if $g_{iq} = p$ then N1$:= N1 \cup \{i\}$;
                    for j$:= 1$ to n do if $g_{jp} = q$ then N2$:= N2 \cup \{j\}$;
                    for every (i,j) $\in$ N1 x N2 do
                        if $u_{ij} = u_{ip} + temp + u_{qj}$ then
                            begin
                                S$:= S \cup \{(i,j)\}$;
                                $v_{ij} := u_{ij}$; $v_{ji} := v_{ij}$; $u_{ij} := d_{ij}$; $u_{ji} := u_{ij}$
                            end; {loop - 1}
                    for h$:= 1$ to n do
                        for every (i,j) $\in$ S do
                            if $u_{ih} + u_{hj} < u_{ij}$ then
                                begin
                                    $u_{ij} := u_{ih} + u_{hj}$; $u_{ji} := u_{ij}$
                                end; {loop - 2}
                    weight$:= 0$;
                    for every (i,j) $\in$ S do
                        begin
                        if $u_{ij} - v_{ij} > \beta_{ij}$ then
                            begin
                                weight$:= weight + r_{ij} \cdot (u_{ij} - v_{ij} - \beta_{ij})$;
                                $\beta_{ij} := u_{ij} - v_{ij}$; $\beta_{ji} := \beta_{ij}$;
                            end,
                                $u_{ij} := v_{ij}$; $u_{ji} := u_{ij}$
                        end; {loop-3}
                    $\mu_k^3 :=$ weight; $d_{pq} :=$ temp; $d_{qp} := d_{pq}$
                end
        end
end;
```

2.3.3 <u>The Lower Plane LP2</u>:

We now derive another lower plane for the NDP with $O(n^3)$ worst case complexity. The following additional numbers are defined for each (i,j) ε Q:

P_{ij} = the set of arcs in the shortest path between the nodes i and j, $(i \neq j)$,

p_{ij} = $|P_{ij}|$,

v_{ij} = the second shortest path length between the nodes i and j,

$\triangle_{ij}$ = $v_{ij} - u_{ij}$,

$$\alpha_{ij}^k = \begin{cases} 1, & \text{if } P_{ij} \text{ contains the arc } a_k \\ 0, & \text{otherwise, } \forall \ k = 1,\ldots, m. \end{cases} \quad (1)$$

$$w_{ij}^k = \triangle_{ij} \ \alpha_{ij}^k / p_{ij}, \ \forall \ k = 1,\ldots, m. \quad (2)$$

<u>Proposition 5</u>: For every pair (i,j) ε Q, $\pi(a_k, (i,j)) \geq \triangle_{ij}$, $\forall \ a_k$ ε P_{ij}.

<u>Theorem 3</u>: $h^4(x) = \mu_o + \mu^4 x$ is a lower plane for the NDP, where $\mu_k^4 = \sum_{(i,j) \ ε \ Q} r_{ij} \cdot w_{ij}^k$

<u>Proof</u>: Consider any pair (i,j) ε Q. It follows from Proposition 5 and Proposition 1 that

$$\triangle_{ij} \ x_k \leq \pi(a_k, (i,j)) \ x_k \leq \pi(X, (i,j)), \ \forall \ a_k \ ε \ P_{ij}. \quad (3)$$

Summing the inequalities in (3) and using (1), we get

$$\sum_{a_k ε P_{ij}} \triangle_{ij} \cdot x_k = \sum_{k=1}^{m} \triangle_{ij} \cdot x_k \alpha_{ij}^k \leq p_{ij} \ \pi(X, (i,j)),$$

which in view of (2) becomes

$$\sum_{k=1}^{m} w_{ij}^{k} x_{k} \leq \pi(X, (i,j)).$$

Multiplying this by r_{ij} and summing for all $(i,j) \varepsilon Q$, we get

$$\sum_{(i,j)\varepsilon Q} r_{ij} \left(\sum_{k=1}^{m} w_{ij}^{k} x_{k} \right) \leq \sum_{(i,j)\varepsilon Q} r_{ij} \pi(X, (i,j))$$

Rearranging terms in LHS and using Proposition 2 in RHS yields

$$\sum_{k=1}^{m} \left(\sum_{(i,j)\varepsilon Q} r_{ij} w_{ij}^{k} \right) x_{k} \leq L(\tilde{X} - X, Q) - \mu_{o}$$

or

$$\mu_{o} + \mu^{4} x \leq L(\tilde{X} - X, Q).$$

A procedure, named LP2, to determine $h^{4}(x)$ is given below.

The procedure computes the first and second shortest path lengths using Minieka's algorithm [21]. Then, each node pair in Q is considered one by one, p_{ij} is computed and an appropriate weight is added to each $a_{k} \varepsilon P_{ij}$.

Proposition 6: The computational complexity of determining $h^{4}(x)$ is $O(n^{3})$.

Proof: The u_{ij}'s and v_{ij}'s can be determined in $O(4n^{3})$ computations [21]. The shortest path between any pair of nodes contains atmost $n - 1$ arcs, hence $O(n)$ computations are performed in each execution of loop - 1. Since this loop is executed $n(n-1)/2$ times, the complexity of the loop is $O(n^{3})$. The overall complexity of the procedure LP2 is, therefore, $O(n^{3})$.

```
procedure LP2;
begin
    compute u_{ij}'s and the corresponding g_{ij}'s;
    compute v_{ij}'s; set μ_o := 0;
    for k: = 1 to m do μ_k^4 := 0;
    for i: = 1 to n-1 do
        for j: = i+1 to n do
            begin
                μ_o: = μ_o + r_{ij}·u_{ij};
                Δ_{ij}: = v_{ij} - u_{ij};
                p_{ij}: = 0; P_{ij}: = ∅;
                q: = j;
                while q ≠ i do
                    begin
                        p_{ij}: = p_{ij}+1;
                        s: = g_{iq};
                        P_{ij}: = P_{ij} ∪ {(s,q)};
                        q: = s
                    end;
                weight: = Δ_{ij}·r_{ij}/p_{ij};
                for every a_k ε P_{ij} do μ_k^4: = μ_k^4 + weight
            end {loop - 1}
end;
```

2.4 COMPUTATIONAL RESULTS

The lower planes derived in this chapter were coded in FORTRAN-IV and tested on network design problems of different sizes with the aim to assess the sharpness of the lower bounds provided and to estimate the computational time requirements. Hoang's and Gallo's lower planes were also coded for comparison purposes. We also tested the effectiveness of the lower planes on a special class of network design problems, known as the Optimum Communication Spanning Tree (OCST) [1,13]. In the OCST problem, a maximum spanning tree problem is solved to obtain the lower bound value after determining μ_k's.

The network design problems were generated randomly with assured connectedness. The distances were considered to be Euclidean in nature and r_{ij}'s were uniformly generated random numbers in the range (1, 100). We considered 3 problem sizes ranging from 25 nodes and 50 arcs, to 50 nodes and 100 arcs, and for each problem size 4 different problems were solved with different number of arcs fixed. Out of all the arcs fixed, approximately 50 percent were included in the network and the rest excluded.

In Gallo's method as well as in LP1, the lower bound value depends upon the order in which arcs are examined for deletion. While implementing Gallo's method, we considered the arcs in the non-increasing order of the c_j values, whereas in LP1 arcs were

considered in the nondecreasing order of the number of shortest paths between all pairs of nodes using the arc.

For each problem considered, the lower bounds were obtained using Hoang's, Gallo's, LP1 and LP2 lower planes. The lower bound value is expressed as (lower bound - μ_o) $100/\mu_o$. We also noted the computational times taken by these methods. These results are presented in Table 2.1 for the NDP and in Table 2.2 for the OCST.

It is apparent from the tables that LP1 yields lower bound values which are generally 1 percent to 3 percent tighter than those of Gallo. Even this much improvement is significant in the network design algorithms and can curtail the implicit enumeration substantially. Both LP1 and Gallo's method yield much sharper bounds than Hoang's method, which is consistent with the findings of Gallo [9]. Further, the computational times of LP1 are generally 20 percent to 40 percent of those taken by Gallo's method. It may, however, be pointed out that computational times taken by Gallo's method can be reduced by incorporating reoptimization procedures not used by us.

The results obtained by LP2 are also encouraging. This method takes times which are closer to those taken by Hoang's method, but bounds provided are much tighter. Our preliminary investigations with the OCST algorithm suggest that use of this plane is justified for moderately small sized network design problems (upto 25 nodes and 50 arcs).

Table 2.1: Comparison of various lower planes for the NDP
(Computational times are in seconds on DEC-1090 system)

Problem No.	No. of nodes	No. of arcs	Budget level B*	No. of fixed arcs	Sharpness of lower bounds				Computational times			
					Hoang	Gallo	LP1	LP2	Hoang	Gallo	LP1	LP2
1	25	50	50%	0	1.40	7.33	10.03	4.41	0.1	1.2	0.5	0.1
2				10	1.39	8.24	9.53	4.07	<0.1	1.1	0.5	0.1
3				20	0.87	6.19	9.10	3.24	<0.1	0.9	0.4	0.1
4				30	0.75	6.52	7.24	2.57	<0.1	0.8	0.3	0.1
5	25	100	50%	0	0.63	5.21	6.81	2.69	0.6	15.6	4.0	0.9
6				20	0.49	4.50	6.16	2.34	0.4	14.1	3.8	0.9
7				40	0.39	4.26	5.47	2.24	0.3	13.2	3.6	0.9
8				60	0.36	6.51	7.45	2.34	0.2	10.8	3.0	0.8
9	50	100	40%	0	1.05	8.03	9.70	3.71	0.6	15.6	4.0	0.9
10				20	0.83	7.53	9.73	3.59	0.4	14.0	3.8	0.9
11				40	0.79	9.32	9.93	3.83	0.3	13.2	3.6	0.9
12				60	1.04	17.54	18.41	4.99	0.2	10.8	3.0	0.8

* $b = \dfrac{B}{100} \sum_{j=1}^{m} c_j$

Table 2.2: Comparison of various lower planes for OCST

(Computational times are in seconds on DEC 1090 System)

Problem No.	No. of nodes	No. of arcs	No. of fixed arcs	Sharpness of lower bounds				Computational times			
				Hoang	Gallo	LP1	LP2	Hoang	Gallo	LP1	LP2
1	25	50	0	2.08	7.10	8.37	5.07	0.1	1.1	0.5	0.1
2			10	2.09	8.84	9.51	5.88	<0.1	1.0	0.4	0.1
3			20	1.95	12.29	13.35	6.61	<0.1	0.9	0.3	0.1
4			32	1.00	9.48	11.03	5.85	<0.1	0.6	0.3	0.1
5	25	50	0	4.75	10.45	11.81	7.18	0.3	1.7	0.4	0.1
6			10	5.57	12.49	14.28	8.29	0.2	1.6	0.4	0.1
7			21	6.33	15.26	16.91	10.20	0.2	1.5	0.3	0.1
8			34	6.77	16.11	18.60	10.80	0.2	1.4	0.3	0.1
9	50	100	0	0.96	7.51	9.72	4.34	0.6	14.6	4.0	0.9
10			21	1.20	10.05	11.19	5.68	0.5	12.7	3.2	0.9
11			42	1.22	11.76	13.97	5.90	0.3	10.9	2.9	0.8
12			61	0.91	11.20	12.59	5.75	0.2	8.5	2.6	0.8

CHAPTER III

EXACT AND HEURISTIC ALGORITHMS FOR OPTIMUM COMMUNICATION SPANNING TREE PROBLEM

3.1 INTRODUCTION:

In this chapter, we present the exact and heuristic algorithms for the Optimum Communication Spanning Tree (OCST) problem. Given an undirected network and the amount of communication r_{ij} between each pair of nodes i and j, the OCST problem is to determine a spanning tree for which the total cost of communication between all pairs of nodes is minimum among all spanning trees. The cost of communication of a given spanning tree is defined as follows: for a pair of nodes i and j, there is a unique path in the spanning tree between i and j. The length of the path is the sum of lengths of arcs in the path. The cost of communication for the pair of nodes i and j is r_{ij} multiplied by the length of the path. Summing over all $n(n-1)/2$ pairs of nodes we have the cost of the spanning tree.

In Section 2, we present the notations used in this chapter. In Section 3, an exact algorithm based on branch and bound approach is presented. In this algorithm, the $O(n^4)$ new lower plane developed in Chapter II is used to obtain the lower bounds. In Section 4, we present two-phase heuristic algorithm for solving the

OCST problem. In Section 5, another heuristic algorithm, named Global heuristic, which is an adaptation of the heuristic algorithm proposed by Dionne and Florian [5] for the Network Design Problem, is given. Dionne and Florian's algorithm was slightly modified for applying it to OCST problem. In Section 6, the detailed computational performance of these three algorithms along with the networks for testing these algorithms is presented.

3.2 NOTATIONS:

Besides the notations used earlier, the following additional notations are used in this chapter.

S = set of nodes which are included in the spanning sub-graph,

$\bar{S}$ = $N - S$

R = set of retained pairs

u_{ij}^O = shortest path distance between the pair i and j, (i ε S, j ε S), over the spanning sub-graph,

z^* = incumbent objective function value,

LB = lower bound,

$$w_i = \begin{cases} \sum_{j\varepsilon\bar{S}} r_{ij}, & \text{if } i \varepsilon S \text{ (amount of communication of node i with nodes in } \bar{S}) \\ \sum_{j\varepsilon S} r_{ij}, & \text{if } i \varepsilon \bar{S} \text{ (amount of communication of node i with nodes in S)} \end{cases}$$

$$\text{wtotal} = \sum_{i\varepsilon S} \sum_{j\varepsilon\bar{S}} r_{ij} = \text{total amount of communication over } (S, \bar{S})$$

$$h_i = \begin{cases} \sum_{j\varepsilon S} w_j \cdot u_{ij}^O, & \text{if } i \varepsilon S \\ \sum_{j\varepsilon\bar{S}} w_j \cdot u_{ij}, & \text{if } i \varepsilon \bar{S} \end{cases}$$

I = set of arcs included in the spanning subgraph,

E = set of arcs to be excluded from the spanning subgraph,

U = A − I ∪ E

= set of arcs unassigned.

As already mentioned S is the set of nodes which are included in the spanning subgraph and it constitutes a connected component with the arcs in the set I. The shortest path distances between pairs (i,j), i ε S and j ε S over the arcs in the sub-graph is given by the u^o matrix.

3.3 THE BRANCH AND BOUND ALGORITHM:

The branch and bound algorithm is essentially an implicit enumeration algorithm. This algorithm proceeds by partitioning the set of all solutions into subsets (called sub problems) which are of reduced size and are progressively easier to solve. An arc is selected and is considered as either included or excluded in the spanning tree thereby generating two candidate problems. The arc selected is known as the branching arc. A lower bound is calculated for the solutions within each subset. The lower bound gives the minimum possible value of the objective function of a feasible solution in that subset. The best solution available at hand at any stage is called the incumbent solution. If for a sub problem, the lower bound exceeds the incumbent value, that sub problem is eliminated from further consideration. This process is known as pruning. If a feasible solution is obtained

then the incumbent is updated, if necessary, and further branching is stopped for this subset. This is known as <u>fathoming</u>. If the sub problem is infeasible then also the subset is eliminated from further consideration. The partitioning process continues until all subsets are pruned or fathomed. Now, we discuss in detail various strategies employed in the branch and bound algorithm for the OCST problem. Fig. 3.1 contains a flow chart of the branch and bound algorithm, a more detailed discussion of each of the steps follows.

<u>Step 1 (Solving the Shortest Path Problem)</u>: The all-pair shortest path problem is solved using Floyd's algorithm over the graph $G(N, A)$.

<u>Steps 2,3,10 (Sub problem Selection)</u>: The stack is a list of sub-problems yet to be examined by the algorithm. Initially, the stack contains a single subproblem which corresponds to the OCST problem. New sub problems are added at the top of the stack. Finally, when the stack is empty, all sub problems have been examined and the incumbent solution is the optimal of OCST problem. At the intermediate stages there exist various strategies for selecting a new sub problem from the stack. The most appealing strategy is the <u>depth first rule</u> which selects the new sub problem as the one that was most recently added to the stack. This approach results in significant reduction in storage, yields a good feasible solution early in the search, makes premature termination of the algorithm attractive and allows reoptimization of sub problems
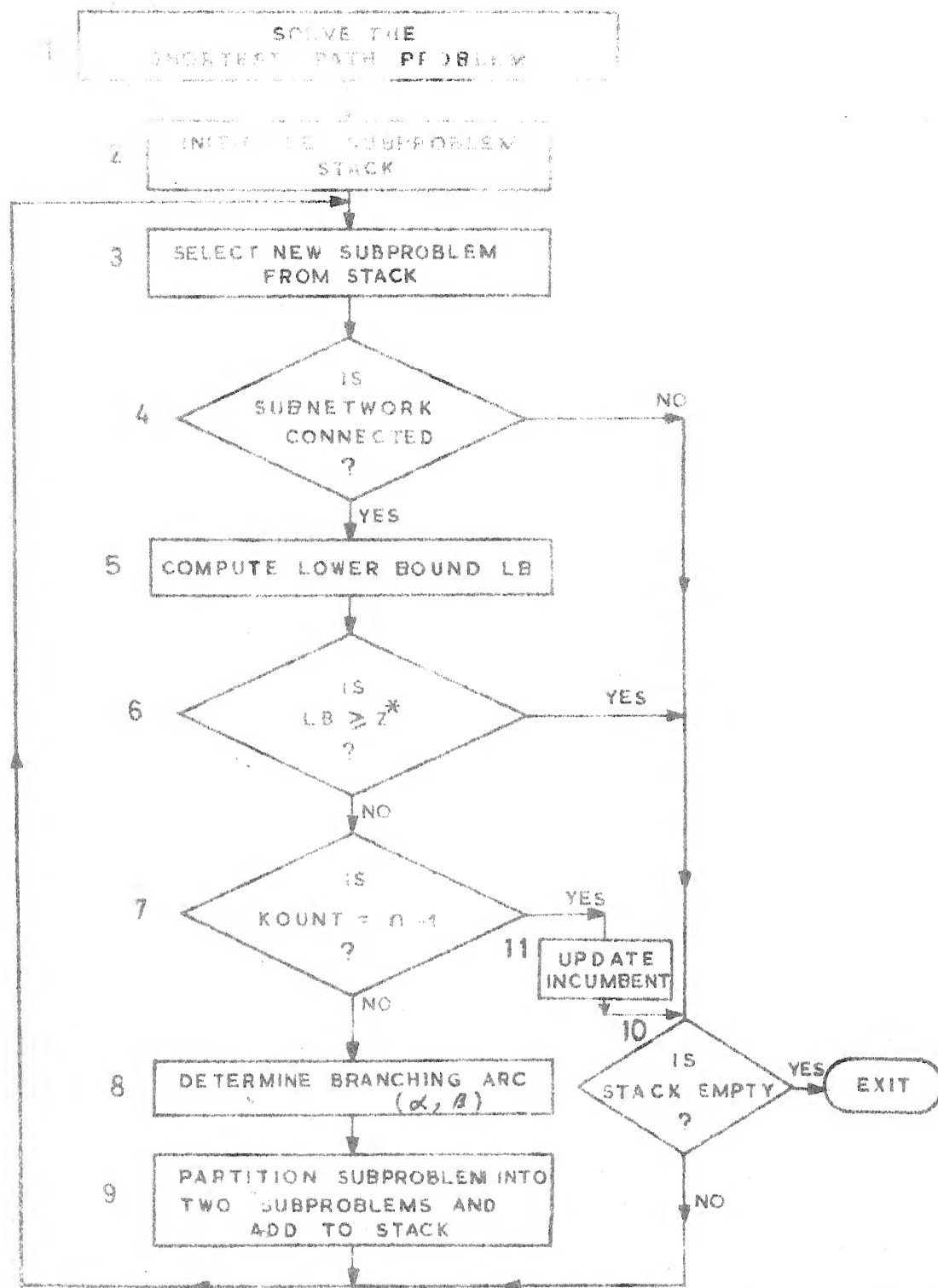
FIG. 3.1 FLOW CHART FOR THE BRANCH AND BOUND ALGORITHM.

In view of these advantages we implemented depth first rule in the algorithm.

<u>Steps 4 and 5 (Lower Bounding)</u>: The lower plane $h^3(x)$ developed in Section 2.3 is used for computing the lower bound for OCST problem. From Section 2.3, we have

$$L(\tilde{A}, Q) + \mu^3 x \leq L(\tilde{A} - X, Q)$$

To get a lower bound, we have to solve the problem

$$\text{Min } L(\tilde{A}, Q) + \mu^3 x$$

such that $G(N, \tilde{A} - X)$ is connected. Equivalently,

$$\text{Min } \sum_{k=1}^{m} \mu_k^3 \cdot x_k \tag{3.1}$$

$$G(N, \tilde{A} - X) \text{ is connected,} \tag{3.2}$$

Define,

$$Y = \tilde{A} - X, \qquad Y \subseteq \tilde{A}$$

$$y_k = \begin{cases} 1, & \text{if } x_k = 0 \\ 0, & \text{if } x_k = 1 \end{cases}$$

$$= 1 - x_k$$

(3.1) and (3.2) becomes

$$\text{Min } \sum_{k=1}^{m} \mu_k^3 \cdot (1 - y_k)$$

$$G(N, Y) \text{ connected.}$$

This reduces to,

$$\text{Min } \left(\sum_{k=1}^{m} \mu_k^3 - \sum_{k=1}^{m} \mu_k^3 \cdot y_k \right) \tag{3.3}$$

$$G(N, Y) \text{ connected.} \tag{3.4}$$

Thus, we have to find $Y \subseteq \tilde{A}$ which satisfies (3.3) and (3.4). This obviously corresponds to a maximum spanning tree (MST) problem, which can be solved using Kruskal's [15] algorithm.

A maximum spanning tree problem is solved with μ^3 taken as edge weights. Let T be the resulting spanning tree. Then the lower bound LB is given by

$$LB = \mu_0 + \sum_{k=1}^{m} \mu_k^3 - \sum_{a_k \varepsilon T} \mu_k^3$$

The procedure LOWERBOUND which computes the lower bound LB is given. The variable totwgt corresponds to $\sum_{k=1}^{m} \mu_k^3$ and tweigt to $\sum_{a_k \varepsilon T} \mu_k^3$. Heap sort method is employed for forming the maximum spanning tree.

In computing the term μ_0, max. (u_{ij}^0, u_{ij}) is used. This is because for the pairs (i,j), $i \varepsilon S$ and $j \varepsilon S$ the shortest path between i and j will be passing through the arcs included in the tree, in the final spanning tree. The shortest path distance for these pairs is given by the u^0 matrix and for the other pairs u^0 matrix contains zero. Since we are using u^0 terms in computing μ_0, β_{ij} is initialized to max. $(u_{ij}^0, u_{ij}) - u_{ij}$ to take care of the increased value.

Step 6 (Pruning Test): After determining the lower bound value, an attempt is made to prune the subproblem, that is, to remove it from further consideration. The pruning test consists of checking $LB > Z^*$ and Z^* is the incumbent value i.e., the objective

```
Procedure LOWERBOUND;
begin
    compute μ₀ := ½ Σ Σ max. (u°ᵢⱼ, uᵢⱼ)·rᵢⱼ;
                     i=1 j=1
    set βᵢⱼ := max. {u°ᵢⱼ, uᵢⱼ}-uᵢⱼ; ∀ i:=1,.., n, ∀ j:= 1,..., n;
    kount := |I|; totwgt := 0;
    for every aₖ ε U do
        begin
            p:= iₖ; q := Jₖ;
            if u_pq = d_pq then
                begin
                    kount :=  kount + 1;
                    temp := d_pq; d_pq := ∞, d_qp := d_pq;
                    N1:= ∅; N2:= ∅; R:= ∅;
                    for i:= 1 to n do if u_iq = u_ip + temp then N1:=N1 U {i};
                    for j:= 1 to n do if u_pj = temp + u_qj then N2:= N2 U {j};
                    for every (i,J) ε N1 x N2 do
                        if uᵢⱼ = u_ip + temp + u_qj then
                            begin
                                R:= R U {(i,J)};
                                vᵢⱼ := uᵢⱼ; vⱼᵢ:= vᵢⱼ; uᵢⱼ:= dᵢⱼ; uⱼᵢ:= uᵢⱼ
                            end;
                    for h:= 1 to n do
                        for every (ı,J) ε R do
                            if u_ıh + u_hJ < uᵢⱼ then
                                begin
                                    uᵢⱼ:= u_ıh + u_hJ; uⱼı:= uᵢⱼ
                                end;
                    cost:= 0; weight := 0;
                    for every (i,J) ε R do
                        begin
                            cost:= cost + rᵢⱼ·(uᵢⱼ-vᵢⱼ);
                            if uᵢⱼ-vᵢⱼ > βᵢⱼ then
                                begin
                                    weight:= weight + rᵢⱼ·(uᵢⱼ-vᵢⱼ-βᵢⱼ);
                                    βᵢⱼ:= uᵢⱼ-vᵢⱼ; βⱼᵢ:= βᵢⱼ
                                end;
                            uᵢⱼ:= vᵢⱼ; uⱼᵢ:= uᵢⱼ
                        end;
```

```
        μ_k := weight; totwgt := totwgt ÷ weight;
        gcost_k := cost;
        d_pq := temp; d_qp := d_pq
      end
   end;
find the maximum spanning tree weight, tweight with μ as
edge weights using Kruskal's algorithm;
LB := μ_0 ÷ (totwgt -tweigt)
end;
```

function value of the best feasible solution enumerated so far.
If the inequality holds, then the sub problem is pruned, else
this sub problem is partitioned into two new sub problems.

Step 7 (Feasibility Test): The testing for feasibility can be
easily incorporated in the LOWERBOUND procedure. If only n-1
arcs are used in the shortest path calculations between all
pairs of nodes in the sub problem, then the sub problem is
clearly feasible to the OCST problem as it forms a spanning
tree. An arc is used in the shortest path calculation when
the shortest distance between the pair incident to that arc is
equal to the length of the arc. The variable 'kount' gives
the number of arcs used in the sub network. When this 'kount'
equals n-1, then the solution value is compared with the current
incumbent value and if found better made the new incumbent.

Steps 8 - 10 (Partitioning): After the pruning test fails, the
sub problem, say P, is partitioned into two new sub problems.
To accomplish this, we select a separation arc (α, β) to parti-
tion the solution set of P into subsets. The global cost of an
arc is defined as the increase in the $\sum_{\forall (i,j) \in Q} u_{ij} \cdot r_{ij}$ value
when an arc is deleted from the network. This is computed in
the procedure LOWERBOUND and stored in array gcost. The branch-
ing arc (α, β) is selected such that it has the maximum global
cost and which connects the already formed subgraph. This
branching arc criteria makes pruning effective when the arc is
excluded from the network as it has maximum global cost. The

procedure BRANCHINGARC which selects the branching arc (α, β), with $\alpha \ \varepsilon \ S$ and $\beta \ \varepsilon \ \bar{S}$ is given.

Procedure BRANCHINGARC;

begin

  mcost: $= -\infty$;

  for every $a_k \ \varepsilon \ U$ do

    if mcost $<$ gcost$_k$ then

      begin

        if $i_k \ \varepsilon \ S$ and $j_k \ \varepsilon \ \bar{S}$ then

          begin

            $\alpha := i_k$; $\beta := j_k$; mcost:= gcost$_k$

          end

        else if $i_k \ \varepsilon \ \bar{S}$ and $j_k \ \varepsilon \ S$ then

            begin

              $\alpha := j_k$; $\beta := i_k$; mcost:= gcost$_k$

            end

      end

end;

The branching arc (α, β) is first considered as included in the spanning tree $(x_{\alpha\beta} = 1)$ and then as excluded from the spanning tree $(x_{\alpha\beta} = 0)$. The two sub problems of P corresponding to $x_{\alpha\beta} = 0$ and $x_{\alpha\beta} = 1$ are added to sub problem stack in the given order. The sub problem with $x_{\alpha\beta} = 1$ remains at the top of the stack.

When an arc (α, β) is included in the spanning tree, all the arcs $[i, \beta]$, $i \in S$ and $i \neq \alpha$, if present, should not be used in the shortest path calculations in the final spanning tree as these arcs form cycles if included in the spanning tree. Thus, for the connected sub-network shown in Fig. 3.2 when arc $[\alpha, \beta]$ is selected for inclusion the arcs shown by dotted lines are to be deleted from the network. These arcs are deleted from the network and the shortest path distances are modified using the procedure DELETEARCS given in Section 1.2. The u^o matrix needs to be updated when an arc (α, β) is added. For all pairs (i, β), $i \in S$, $u^o_{i\beta}$ and $u^o_{\beta i}$ are set equal to $u^o_{i\alpha} + d_{\alpha\beta}$. Similarly while back tracking the arcs are added to the network and u, u^o matrices are modified accordingly.

<u>Numerical Example</u>:

A numerical example is solved to illustrate the methodology of the branch and bound algorithm. The network contains 7 nodes and 13 arcs. The distance matrix d and the requirement matrix r are given. The example network is shown in Fig. 3.3. The branching tree enumerated is given in Fig. 3.5. Nodes are numbered by the order in which they are emunerated. The lower bound obtained at each node is written in a rectangle at that node. The left branching indicates inclusion of an arc in the spanning tree and the right branching indicates the exclusion of that arc. The nodes incident to the branching arc is also shown. The Z^* is initialized to the heuristic solution value of 91004. The seed node is taken as 4.
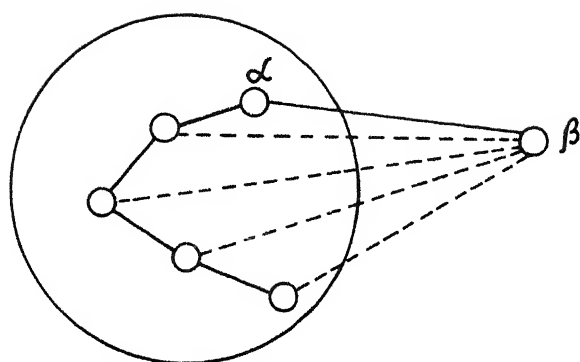
FIG. 3.2
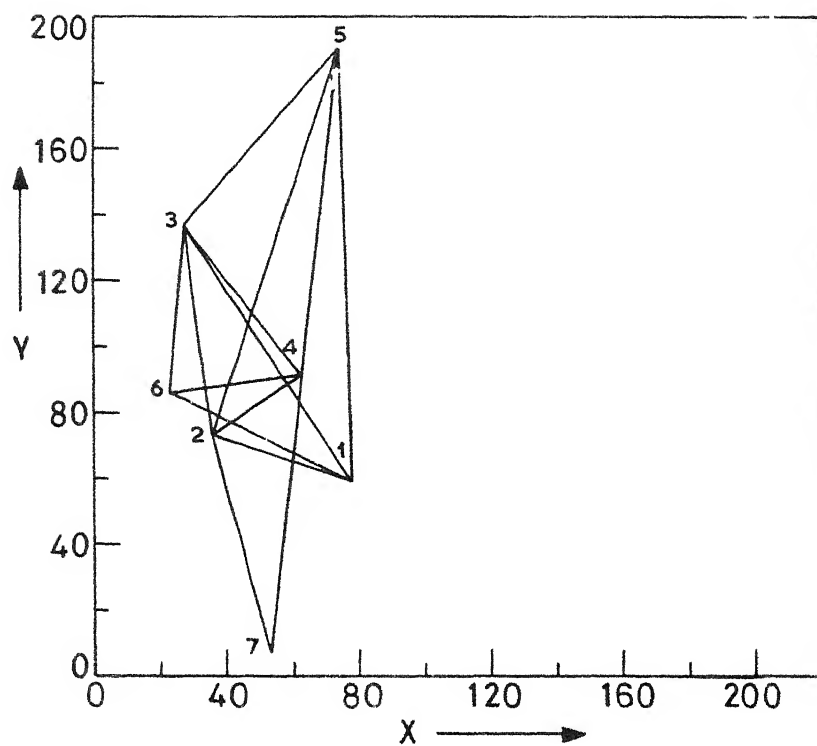
FIG.3.3   EXAMPLE NETWORK
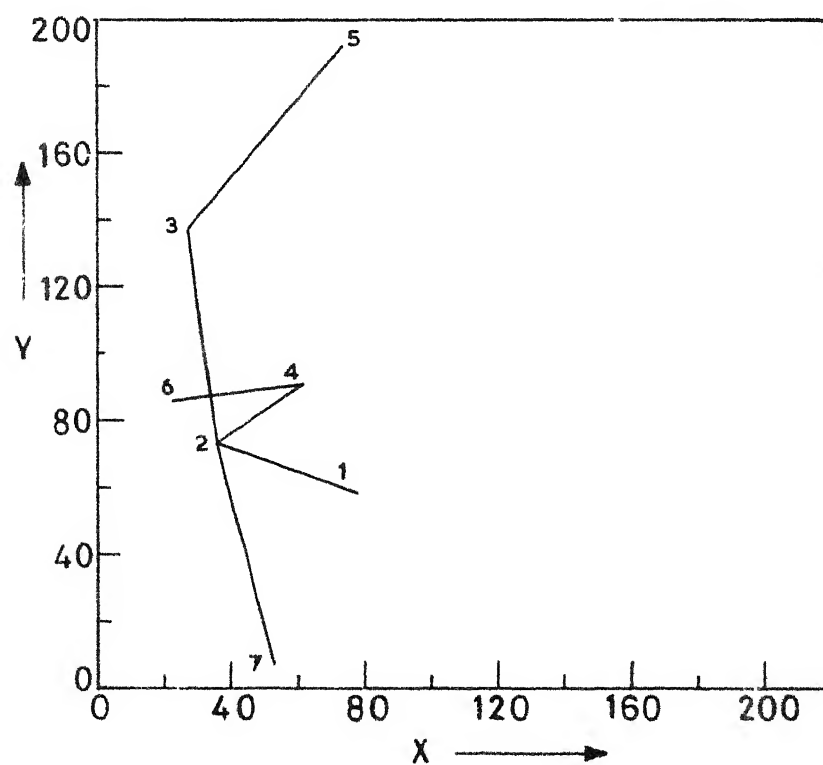
FIG.3.4   TWO   PHASE   HEURISTIC SOLUTION.

FIG. 3.5

$$
d \qquad\qquad\qquad\qquad\qquad\qquad\qquad r
$$

$$
\begin{bmatrix}
0 & 44 & 93 & \infty & 133 & 61 & \infty \\
44 & 0 & 64 & 30 & 124 & \infty & 68 \\
93 & 64 & 0 & 57 & 72 & 51 & \infty \\
\infty & 30 & 57 & 0 & \infty & 38 & \infty \\
133 & 124 & 72 & \infty & 0 & \infty & 186 \\
61 & \infty & 51 & 38 & \infty & 0 & \infty \\
\infty & 68 & \infty & \infty & 186 & \infty & 0
\end{bmatrix}
\qquad
\begin{bmatrix}
0 & 20 & 74 & 61 & 33 & 11 & 19 \\
20 & 0 & 7 & 72 & 64 & 5 & 68 \\
74 & 7 & 0 & 40 & 38 & 54 & 89 \\
61 & 72 & 40 & 0 & 41 & 80 & 44 \\
33 & 64 & 38 & 41 & 0 & 2 & 14 \\
11 & 5 & 54 & 80 & 2 & 0 & 68 \\
19 & 68 & 89 & 44 & 14 & 68 & 0
\end{bmatrix}
$$

3.4 TWO-PHASE HEURISTIC ALGORITHM:

The two-phase heuristic algorithm consists of two phases:

(1) Tree-building phase, and

(2) Tree-improvement phase.

The tree-building algorithm starts with a seed node and successively builds the tree by including one arc each time. The arc selected was such that it yields a very good feasible solution. The method includes that arc in the tree, whose one end belongs to the tree already formed (i.e., belongs to S) and the other end belongs to $\bar{S}$ The amount of communication of a node i with all nodes in the other set is given by w_i. The total communication of nodes in set S with nodes in set $\bar{S}$ is given by wtotal. Labels (h) are associated with each node as given in Section 3.2 and the arc is selected for which $h_i + wtotal.d_{ij} + h_j$ is minimum among all arcs (i,j) such that i ε S and j ε $\bar{S}$. The procedure SELECTARC which finds the arc

to be included in the spanning tree is given. This method terminates when a spanning tree is formed with n-1 arcs.

In procedure SELECTARC, all the setting operations are of $O(n^2)$ complexity and the complexity of loop-1 is also $O(n^2)$. Thus the complexity of procedure SELECTARC is $O(n^2)$.

In procedure TREEBUILDING the complexity of determining u_{ij}'s is $O(n^3)$ using Floyd's algorithm. $O(n)$ operations are performed in loop-1. Since SELECTARC has $O(n^2)$ complexity and as loop-2 is executed n-1 times, the complexity of TREE-BUILDING is $O(n^3)$. The arrays tail and head contain the end nodes of the arcs of the spanning tree and the variable tcost the associated cost of the tree.

Tree Improvement Algorithm:

The spanning tree thus constructed by the tree-building algorithm is fed to the tree-improvement algorithm which examines each tree arc one-by-one and tries to interchange it with an appropriate non-tree arc so that the cost of communication decreases. Whenever a profitable interchange is found, it is made and the process is repeated with the changed spanning tree. The algorithm terminates when no possible interchange yields improvement. Each tree-arc [s,t] is considered one-by-one and assuming that the arc is deleted, the two sets of nodes S and $\bar{S}$ are formed. Labels are associated with each node as defined in Section 3.2. For each arc in the cutset $(S, \bar{S})$, the

```
Procedure SELECTARC;
begin
```

$$
\text{compute } w_i := \begin{cases} \sum_{j \in \bar{S}} r_{ij}, & \text{if } i \in S \\ \sum_{j \in S} r_{ij}, & \text{if } i \in \bar{S} \end{cases} \quad , \ \forall \ i := 1, \ldots, n;
$$

$$
\text{wtotal} := \sum_{i \in S} \sum_{j \in \bar{S}} r_{ij};
$$

$$
h_i := \begin{cases} \sum_{j \in S} w_j \cdot u^O_{ij}, & \text{if } i \in S \\ \sum_{j \in \bar{S}} w_j \cdot u_{ij}, & \text{if } i \in \bar{S} \end{cases} \quad \forall \ i := 1, \ldots, n;
$$

```
    mcost := ∞;
    for every [i,j] ε S x S̄ do
      begin
        cost:= h_i + wtotal . d_ij + h_j;
        if cost < mcost then
          begin
            p:= i; q:= j; mcost:= cost
          end
      end {loop-1}
end;
Procedure TREEBUILDING (seed);
begin
  determine all pair shortest path lengths u_ij's,
  set s:= seed; S:= {s}; S̄:= N-{s}; u°_ss:= 0;
  for k:= 1 to n-1 do
    begin
      select the arc to be added [p,q] using SELECTARC;
      for every i ε S do
        begin
          u°_iq:= u°_ip + d_pq; u°_qi:= u°_iq
        end; {loop-1}
      tail [k]:= p; head [k]:= q; S:=S U {q}; S̄:= S̄ - {q}
    end; {loop-2}
```

$$
\text{set tcost} := \frac{1}{2} \sum_{i=1}^{n} \sum_{j=1}^{n} r_{ij} \cdot u^O_{ij};
$$

```
end;
```

cost of communication if that arc is exchanged with the arc [s,t] is found and the arc [p,q] with minimum cost is obtained. If the arc [p,q] decreases the total cost of communication then an exchange is made otherwise another tree-arc is selected and the process terminates when all tree-arcs are considered for exchange and none yields improvement. The algorithm is stated below as procedure TREEIMPROVEMENT.

In procedure TREEIMPROVEMENT, loop-1 has $O(n)$ complexity. The setting operations require $O(n^2)$ computations and loop-2 also has $O(n^2)$ complexity. Since loop-3 is performed at most $(n-1)$ times for each profitable exchange, in total $O(n^3)$ computations are performed in loop-3. Thus TREEIMPROVEMENT has $O(n^3)$ complexity for every profitable exchange and the variable _iter_ gives the number of exchanges performed by the procedure.

Numerical Example:

The same example that is considered in Section 3.3 is taken to illustrate the two-phase heuristic algorithm. For the tree-building algorithm, the starting node is taken as 4. The value of w, wtotal, label and the branching arc (p,q) selected at each stage is shown in Table 3.1. The tree-improvement algorithm performed only one iteration. The tree-arc (4,3) is exchanged with the arc (2,3). The solution value obtained by the two-phase heuristic algorithm is 91004. In this case this is also an optimum solution. The solution obtained by the heuristic algorithm is shown in Fig. 3.4.

```
Procedure TREEIMPROVEMENT;
begin
   iter:= 0; k:= 0;
   100 : iter:= iter+1; count := 0;
   repeat
      count: = count + 1;
      k:= k mod (n-1); k:= k+1;
      s:= tail [k]; t:= head [k]; S:= ∅; S̄ := ∅;
      for i:= 1 to n do if uᵢₛᵒ + dₛₜ = uᵢₜᵒ then S: = S ∪ {i}
                         else S̄:= S̄ ∪ {i}; {loop-1}
```

$$w_{ii} := \begin{cases} \sum_{j \in \bar{S}} r_{ij}, & \text{if } i \in S \\ \sum_{j \in S} r_{ij}, & \text{if } i \in \bar{S} \end{cases} \quad \forall\, i = 1, \ldots, n;$$

$$wtotal := \sum_{i \in S} \sum_{j \in \bar{S}} r_{ij};$$

$$h_i := \begin{cases} \sum_{j \in S} w_j \cdot u_{ij}^o, & \text{if } i \in S \\ \sum_{j \in \bar{S}} w_j \cdot u_{ij}^o, & \text{if } i \in \bar{S} \end{cases} \quad \forall\, i = 1, \ldots, n;$$

```
      pcost:= hₛ + wtotal. dₛₜ + hₜ; mcost:= ∞;
      for every [i,j] ε (S x S̄)∩ A do
         begin
            cost:= hᵢ + wtotal. dᵢⱼ+hⱼ;
            if mcost > cost then
               begin
                  p:= i; q: = j; mcost: = cost
               end
         end; {loop-2}
   until (mcost < pcost) or (count = n-1); {loop-3}
   if mcost < pcost then
      begin
         tcost:= tcost - (pcost - mcost);
         tail [k]:= p; head [k]: = q;
         for every (i,j) ε (S x S̄) do uᵢⱼᵒ:= uᵢₚᵒ + dₚ𝔮 + u𝔮ⱼᵒ;
         goto 100
      end
end;
```

Table 3.1

Node No.	w							wtotal	h							Arc (p,q)
	1	2	3	4	5	6	7		1	2	3	4	5	6	7	
1	61	72	40	338	41	80	44	338	22149	18760	23121	0	37945	21684	35514	(4,2)
2	81	164	47	266	105	85	112	430	36065	7980	34212	4920	45444	35485	46366	(4,6)
3	92	159	101	186	107	135	180	480	43784	14760	40020	9900	52988	17880	43538	(4,3)
4	166	152	201	146	145	81	269	580	49413	27375	29241	19095	72112	34979	45562	(2,1)
5	52	132	127	85	178	70	288	466	36575	20647	29791	17707	53568	30095	33108	(2,7)
6	33	64	38	41	192	2	14	192	12620	7076	14588	7976	0	15120	18228	(3,5)

3.5 GLOBAL HEURISTIC ALGORITHM:

This heuristic is essentially similar to the branch and bound algorithm, except that in computing the lower bound, the global cost of deletion of arcs is considered as the weight of the arc. In this method, each arc is considered one-by-one, and the global cost of deletion of that arc is found and that cost is associated with the arc. This will not be a valid lower bound, as the effect on a pair may be counted more than once. As such there is a possibility of elimination of the optimal solution, but Dionne and Florian [5] have observed that the probability of over looking the optimal solution is very low and an effective heuristic results by considering the above weights in calculating the lower bound in branch and bound algorithm. Since it considers the global cost of deletion of an arc as the weight associated with that arc, it is named as global heuristic. This algorithm is essentially similar to the branch and bound algorithm discussed in Section 3.3, except that the following modifications are made in the LOWERBOUND procedure.

i) μ_0 is computed as $\frac{1}{2} \sum_{i=1}^{n} \sum_{j=1}^{n} u_{ij} \cdot r_{ij}$

ii) β_{ij} is set to 0; $\forall\ i = 1,\ldots, n, \forall\ j = 1,\ldots, n.$

iii) μ_k is set equal to the global cost of deleting arc a_k (i.e., $\mu_k = gcost_k$).

3.6 COMPUTATIONAL INVESTIGATIONS:

In this section, we present the computational performance of the branch and bound algorithm, the two-phase heuristic algorithm and the global heuristic algorithm. All these algorithms are coded in FORTRAN-IV and implemented on DEC 10 system.

The networks considered for the testing of these algorithms are generated randomly with assured connectedness. Two types of networks are considered for the testing of these algorithms. In the first type the distances are considered to be Euclidean in nature. The x and y coordinates of the nodes are generated randomly in a plane. The plane size is taken as (200, 200). Then the required number of arcs are added to the network by generating the tail and head nodes of the arcs. The lengths of arcs are taken as the Euclidean distance between its end nodes. Additional arcs are then added to the network, if necessary, to make the degree of each node atleast 2.

In the second type of networks, the distances are generated randomly in the range (0, 200). The arcs are generated by the same method as given above.

For both types of networks, the r_{ij}'s are generated randomly in the range (1, 100).

The tree building algorithm needs a starting seed node. Four nodes are considered as starting nodes, each node from each quadrant of the plane having maximum total communication

with other nodes. The best solution value among these four is taken as the solution provided by the two-phase heuristic algorithm.

For the branch and bound algorithm the starting node is taken as that node having maximum total amount of communication with other nodes. The lower bound value in the branch and bound algorithm depends upon the order in which the arcs are examined. The arcs are considered in the nondecreasing order of the number of shortest paths between all pairs of nodes using the arc. This ordering was found to be very effective on the number of nodes examined by the branch and bound algorithm. For both the branch and bound algorithm and the global heuristic algorithm the initial incumbent value is taken as the solution value provided by the two-phase heuristic algorithm.

In Table 3.2, we give the computational performance of the branch and bound algorithm and the global heuristic algorithm for Euclidean networks. The number of nodes examined by the algorithm and the maximum depth of the branch and bound tree are given. In this table Z corresponds to the solution value obtained by the global heuristic algorithm and Z^* the optimal solution value obtained by the branch and bound algorithm.

In Table 3.3, the results of the two-phase heuristic algorithm are given for Euclidean networks. The first 10 problems correspond to the 10 problems given in Table 3.2. In this

Table, z^1 is the solution value obtained by the first phase of the heuristic algorithm (tree building algorithm) and z^2 corresponds to the second phase (i.e., tree-improvement algorithm). The number of iterations taken by the tree-improvement algorithm are also given.

The results for the branch and bound algorithm, the global heuristic algorithm and the two-phase heuristic algorithm in the case of random networks are given in Tables 3.4 and 3.5.

The branch and bound algorithm when applied to Euclidean networks is able to solve problems of size 40 nodes and 69 arcs in about 40 minutes of CPU time. The times taken by the algorithm and the nodes examined are found to be exponentially growing with the problem size. The sparser problem appear to be easier to solve than the denser problems as the lower bounding is tighter in such cases. The performance of the branch and bound algorithm is much more encouraging for random networks where problems of size 30 nodes and 54 arcs are solved in less than 30 seconds. We believe that this dramatic difference in performance results due to large number of near-optimum solutions which makes Euclidean problems inherently difficult. The similar observations have been drawn by researchers in solving other combinatorial optimization problem, in particular, the traveling salesman problem.

Table 3.2: Performance of the Branch and Bound Algorithm and the Global Heuristic Algorithm for Euclidean Networks.

Prob-lem No.	No. of Nodes	No. of Arcs	Branch and Bound Algorithm				Global Heuristic Algorithm			
			Z^*	Nodes exami-ned	Max. depth.	Time in Sec.	Z/Z^*	Nodes exami-ned	Max. depth	Time in Sec.
1	10	17	421106	67	10	1.03	1.000	63	10	0.93
2	10	20	337831	150	14	2.03	1.000	1	0	0.04
3	10	30	293664	193	12	3.24	1.000	1	0	0.10
4	20	31	3576257	807	20	104.24	1.000	313	17	39.51
5	20	40	2942480	3571	23	457.54	1.000	1575	23	179.75
6	20	50	1871825	6258	28	643.42	1.000	5787	26	552.61
7	25	50	3491057	434	24	98.08	1.000	231	21	52.07
8	30	52	6378467	1452	34	495.59	1.000	418	34	117.03
9	30	61	5050000	6059	33	2091.46	1.000	1743	32	560.55
10	40	69	11661376	2591	46	2247.58	1.000	433	42	364.12

Table 3.3: Performance of the Two-Phase Heuristic Algorithm for Euclidean Networks.

Problem No.	No. of Nodes	No. of Arcs	Z^1/Z^2	Time in Sec.	Z^2/Z^{2*}	No. of Iter.	Time in Sec.	Total time in Secs.
1	10	17	1.000	0.01	1.000	1	0.01	0.02
2	10	20	1.000	0.01	1.000	1	0.01	0.02
3	10	30	1.000	0.01	1.000	1	0.01	0.02
4	20	31	1.005	0.13	1.000	2	0.11	0.24
5	20	40	1.012	0.11	1.000	3	0.15	0.26
6	20	50	1.001	0.12	1.000	2	0.15	0.27
7	25	50	1.007	0.22	1.000	2	0.24	0.46
8	30	52	1.000	0.38	1.000	1	0.25	0.63
9	30	61	1.001	0.39	1.000	2	0.37	0.76
10	40	69	1.007	0.87	1.000	2	0.62	1.49
11	20	60	1.009	0.12	-	2	0.11	0.23
12	25	60	1.010	0.22	-	3	0.23	0.45
13	30	76	1.024	0.38	-	6	0.60	0.98
14	40	83	1.000	0.86	-	1	0.60	1.46
15	40	100	1.045	0.87	-	8	1.63	2.50
16	50	87	1.002	1.67	-	3	2.29	3.96
17	50	101	1.006	1.66	-	3	1.63	3.29
18	50	200	1.003	1.67	-	6	2.32	3.99
19	75	154	1.001	5.51	-	4	6.55	12.06
20	75	300	1.011	5.56	-	10	7.73	13.29
21	75	600	1.001	5.62	-	3	6.57	12.19
22	100	214	1.001	12.96	-	5	16.83	29.79
23	100	500	1.001	13.13	-	5	24.85	37.98
24	100	1000	1.001	13.25	-	4	13.35	26.60

Table 3.4: Performance of the Branch and Bound Algorithm and the Global Heuristic Algorithm for Random Networks.

Prob-lem No.	No. of Nodes	No. of Arcs	Branch and Bound Algorithm				Global Heuristic Algorithm			
			Z^*	Nodes exami-ned	Max. depth	Time in Sec.	Z/Z^*	Nodes exami-ned	Max. depth	Time in Sec.
1	10	15	299687	18	9	0.22	1.000	18	9	0.21
2	10	20	296140	23	9	0.29	1.000	23	9	0.27
3	10	30	280408	38	9	0.65	1.000	29	8	0.50
4	20	34	2333604	161	21	19.65	1.000	143	23	16.44
5	20	41	1235441	73	19	6.94	1.000	65	19	6.16
6	20	50	1226077	127	19	13.77	1.000	105	18	10.44
7	25	53	2152375	143	23	26.59	1.000	71	16	14.84
8	30	54	5291672	83	29	26.44	1.000	65	30	21.31

Table 3.5: Performance of the Two-Phase Heuristic Algorithm for Random Networks.

Problem No.	No. of Nodes	No. of Arcs	Z^1/Z^2	Time in Sec.	Z^2/Z^*	No. of Iter.	Time in Sec.	Total Time in Secs.
1	10	15	1.000	0.02	1.000	1	0.01	0.03
2	10	20	1.000	0.02	1.000	1	0.01	0.03
3	10	30	1.000	0.03	1.000	1	0.01	0.04
4	20	34	1.000	0.11	1.000	1	0.07	0.18
5	20	41	1.000	0.11	1.000	1	0.08	0.19
6	20	50	1.009	0.11	1.000	4	0.25	0.36
7	25	53	1.002	0.21	1.000	2	0.26	0.47
8	30	54	1.000	0.37	1.000	1	0.26	0.63
9	20	60	1.000	0.11	—	1	0.07	0.18
10	30	61	1.000	0.38	—	1	0.26	0.64
11	30	75	1.000	0.38	—	1	0.25	0.63
12	40	64	1.000	0.86	—	1	0.60	1.46
13	40	81	1.000	0.86	—	1	0.60	1.46
14	40	101	1.001	0.86	—	2	1.02	1.88
15	50	87	1.000	1.67	—	1	1.16	2.83
16	50	107	1.003	1.67	—	2	1.81	3.48
17	50	200	1.003	1.67	—	3	2.04	3.71
18	75	155	1.003	5.53	—	2	7.11	12.64
19	75	300	1.003	5.56	—	5	7.67	13.23
20	75	600	1.000	5.58	—	1	3.87	9.45
21	100	212	1.001	13.00	—	3	14.57	27.57

For both the branch and bound and global heuristic algorithms, the initial incumbent value is taken as the solution obtained by the two-phase heuristic algorithm. No improvement is made by these algorithms as the initial incumbent solution is itself the optimal solution for all the problems solved. All the time is spent in proving the optimality of the solution. This shows the ineffectiveness of the lower bounds available for the network design problems.

We do not observe any significant difference in performance between the branch and bound algorithm and the global heuristic algorithm. Though the computational times taken by the global heuristic algorithm are 1-5 times smaller than the branch and bound algorithm, the growth function is again found to be exponential in nature. Our experience of global heuristic algorithm is in contrast with the experience of Dionne and Florian [5] for the network design problem whose have reportedly obtained 'excellent' results by this method. Our experience indicates global heuristic algorithm can not be used to solve reasonably large sized Euclidean problems. It is worth noting that the lower bounds obtained by the global heuristic algorithm are upper bounds on the lower bounds of any branch and bound algorithms which considers arc deletion one by one. Hence until we are able to consider block deletion of arcs effectively, which will be a major theoretical break through, we can not expect the branch and bound algorithm attractive for solving

larger sized instances of the network design problems and OCST problem in particular. Until then, we shall have to take the recourse to efficient heuristic algorithms.

The two-phase heuristic algorithm developed by us in this thesis is found to be very attractive from the point of view of accuracy and computational time. A good solution is obtained in the tree-building phase, which is then converted into an optimal solution in the tree-improvement phase in all the twenty problems considered by us. The computational times also appear to be very reasonable. For instance, problems of size 100 nodes and 1000 arcs are solved in less than 30 seconds of CPU time. The algorithm spends almost equal time in both the phases. We attribute this encouraging performance of the algorithm to the network topology, the structure of the spanning tree. The tree-building and tree-improvement methods exist to obtain minimum spanning tree. It is therefore reasonable to expect that adaptations of such approaches will yield good solutions for the OCST problem.

We also did some investigations to note the effect of variations in the size of the plane in which the points are generated randomly and the interval from which r_{ij} are drawn randomly. We did not observe any significant change in the performance of any of the algorithm. We also applied the algorithm on random planar networks and similar results are obtained by us.

In passing, we comment on one possible improvement in the branch and bound algorithm. We observed the lower bounding routine to be the slowest portion in the algorithm where it spends about 90% of the time. The routine currently used does not exploit the sparseness of the network. We believe that the computational times of the algorithm can be drastically decreased if an improved method to determine the effect of arc deletion is incorporated. This will allow us to solve larger sized problem instances.

REFERENCES

[1] R.K. Ahuja and V.V.S. Murty, Exact and heuristic algorithms for optimum communication spanning tree, Working paper, Industrial and Management Engineering Programme, Indian Institute of Technology, Kanpur, 1985.

[2] D.E. Boyce, A. Farhi and R. Weischedel, Optimal network problem: a branch and bound algorithm. Environ. Plan. 5(1973) 519-533.

[3] D.E. Boyce and J.L. Soberanes, Solution to the optimal network design problem with shipment related to transportation cost, Trans. Res. 13B (1979) 65-80.

[4] G. Dantzig, R. Harvey, Z. Lansdowne, D. Robinson and S. Maier, Formulating and solving the network design problem by decomposition. Trans. Res. 13B (1979) 5-17.

[5] R. Dionne and M. Florian, Exact and approximate algorithms for optimal network design. Networks 9(1979) 37-59.

[6] M. Florian, G.G. Guerin and G. Bushel, The engine scheduling problem on a railway network. INFOR J. 14 (1976) 121-128.

[7] R.W. Floyd, Algorithm 97: shortest paths. Comm. ACM 5 (1962) 345.

[8] G. Gallo, A new branch and bound algorithm for the network design problem. Report L 81-1, Instituto Di Elaborazione Dell Informazione, Pisa, Italy, 1981.

[9] G. Gallo, Lower planes for the network design problem. Networks 13(1983) 411-426.

[10] A.M. Geoffrin and G. Graves, Multicommodity distribution system design by Benders decomposition. Mgmt. Sci. 5(1974) 822-844.

[11] H.H. Hoang, A computational approach to the selection of an optimal network. Mgmt. Sci. 19(1973) 488-498.

[12] H.H. Hoang, Topological optimization of networks: a nonlinear mixed integer model employing generalized Benders decomposition. IEEE Trans. Automatic Control AC-27 (1983) 164-169.

[13] T.C. Hu, Optimum communication spanning trees. SIAM J. Computing 3 (1974) 188-195.

[14] D.S. Johnson, J.K. Lenstra and A.H.G. Rinnooy Kan, The complexity of the network design problem. Networks 8(1978) 279-285.

[15] J.B. Kruskal, On the shortest spanning subtree g graph and the traveling salesman problem, Proc. Amer. Math. Soc. 7(1956) 48-50.

[16] L.J. Leblanc, An algorithm for the discrete network design problem. Trans. Sci. 9 (1975) 283-287.

[17] M. Los and C. Lardinois, Combinatorial programming, statistical optimization and the optimal transportation network problem. Trans. Res. 16 B (1980) 89-124.

[18] T.L. Magnanti, P. Mireault and R.T. Wong, Tailoring Benders decomposition for network design. Working paper OR 125-83, Operations Research Center, Massachusetts Institute of Technology, 1983.

[19] T.L. Magnanti and R.T. Wong, Accelerating Benders decomposition: algorithmic enhancement and model selection criteria. Opns. Res. 29 (1981) 464-484.

[20] T.L. Magnanti and R.T. Wong, Network design and transportation planning: models and algorithms. Trans. Sci. 18 (1984) 1-55.

[21] E. Minieka, On computing sets of shortest paths in a graph. Comm. ACM 17 (1974) 351-353.

[22] R.Richardson, An optimization approach to routing aircraft. Trans. Sci. 10 (1976) 52-71.

[23] A.J. Scott, The optimal network problem: some computational procedures. Trans. Res. 3 (1969) 201-210.

[24] M.M. Syslo, Narsingh Deo and J.S. Kowalik, Discrete Optimization Algorithms with Pascal Programs. Prentice-Hall Inc., Englewood Cliffs, NJ 07632, 253-259.

Branch and Bound Algorithm for
Optimum Communication Spanning Tree (OCST) Problem

V.V.S.Murty.

Input Data   :

N   :   Number of nodes
M   :   Number of arcs
ISEED   :   Seed value
NGN   :   1   for Euclidean Networks
           :   2   for Random Networks
EPSLON   :   Tolerable % deviation
           :   from Optimum Solution
ZSTAR   :   Incumbent solution value

Important Variables   :

INFTY   :   A large number : 1000000
SEED   :   Starting seed node
NODES   :   Number of nodes examined by
           :   the branch and bound algorithm
TOP   :   Pointer to the top of stack
NIMP   :   Number of improvements made
           :   on the initial incumbent
MDEPTH   :   Maximum depth of the
           :   branch and bound tree
INC   :   Number of arcs included
           :   the connected component
TOTWGT   :   Total weights of all arcs
MSTWGT   :   Weight of the maximum spanning tree
X   :   Starting node of set S
Z   :   Obj.fn.value of a feasible solution
TLIMIT   :   Time limit in Milli Seconds
ORDER(4)   :   Contains 4 nodes supplied
           :   by Network Generator
PARENT(N)   :   SEED if I belongs to S
                       :   -1 otherwise
HEAD(N)   :   Contains the head node of arcs
                       :   of the tree corresponding to
                       :   the incumbent solution
TAIL(N)   :   Contains the corresponding tail node
HD(N)   :   Contains the head node of arc
                       :   of the tree corresponding to
                       :   the feasible solution
TL(N)   :   Contains the corresponding tail node.
LINK(N)   :   Gives the next node in the set S
                       :   INFTY if that is the last node
                       :   0 if the node does not belong to S
                       :   S of branching arc
HNODE(M)   :   Head node of arcs
TNODE(M)   :   Tail node of arcs
LENGTH(M)   :   Contains the length of the arc
STATUS(M)   :   1   if arc is included in the tree
                       :   2   if arc is excluded from the tree
                       :   0   otherwise
IORDER(M)   :   Contains the order of arcs
MU(M+1)   . :   For storing the weights of arcs
                       :   in Lower Bound calculation
ARC(M)   :   The corresponding arc numbers
INODE((N-1)*N/2):   For retaining pairs
JNODE((N-1)*N/2):
V((N-1)*N/2)   :   For storing the corresponding
                       :   shortest path distance
STACK(M,2)   :   For storing the end nodes
                       :   of the branching arc
D(N,N)   :   Distance matrix
R(N,N)   :   Requirement matrix
U(N,N)   :   Shortest path distance matrix
UO(N,N)   :   Shortest path distance matrix

```
      RETAIN(V,V)          :    for pairs belonging to set S
                                For checking whether a pair is
                                retained in that iteration
      BETA(V,V)            :    For storing the maximum shortest
                                path distance considered for a pair
      R1,R2                :    Range of R
      X1,X2                :    Plane size for X coordinate
      Y1,Y2                :    Plane size for Y coordinate
      NODDEG               :    Minimum degree of node
                                that is required    .

      INTEGER I,J,DIK,DPJ,FIRST,P,Q,PP,QQ,
     1        SEED,S,TOP,TEMP,TOT,GF,TCOUNT,TWEIGT,
     2        R1,R2,X1,X2,Y1,Y2,X,Z,ASTAR,WEIGHT,
     3        RSUM,TIME1,TIME2,TFINL,TLIMIT,
     4        ORDER(4),PARENT(50),PARELT(50),
     5        H(50),FL(50),HEAD(50),TAIL(50),
     6        LINK(50),STPT(50),SELF(50),
     7        TNODE(120),MNODE(120),LENGTH(120),
     8        STATUS(120),TTYPE(120),NU(121),ARC(120),
     9        TNODE(1250),UTIL(1250),V(1250),
     1        STACK(120,2),N(50,50),ARCNUM(50,50),
     2        U(50,50),UU(50,50),RETAIN(50,50),
     3        R(50,50),BETA(50,50)
      DATA INFTY,LIMIT,ARCNUM/1000000,3500000,2500*1000000/
      DATA NODDEG,R1,R2,X1,X2,Y1,Y2/2,1,100,0,200,0,200/
      READ(31,*) N,M,ISEED,NGN,EPSLON,ZSTAR
      GO TO (10,20),NGN
10    CALL NG1(N,M,ISEED,NODDEG,R1,R2,X1,X2,Y1,Y2,D,R,ORDER)
      GO TO 30
20    CALL NG2(N,M,ISEED,NODDEG,R1,R2,X1,X2,Y1,Y2,D,R,ORDER)
30    CONTINUE

      Initialization

      NMIN1=N-1
      SFC)=ORDER(1)
      K=0
      TOT=S=0
      CALL RTIME(TIME1)
      TLIMIT=TIME1+LIMIT
      DO 40 I=1,NMIN1
      DO 40 J=I+1,N
      IF(D(I,J).EQ.INFTY) GO TO 40
      K=K+1
      HEAD(K)=I
      TAIL(K)=J
      LENGTH(K)=D(I,J)
      STATUS(K)=J
      NU(K)=0
      ARCNUM(I,J)=K
      ARCNUM(J,I)=K
40    CONTINUE
      DO 50 I=1,N
      LINK(I)=0
      PARENT(I)=-1
      DO 50 J=1,N
      U(I,J)=D(I,J)
      BETA(I,J)=I
      UU(I,J)=0
      RETAIN(I,J)=0
50    BETA(I,J)=0

      Apply FLOYD'S Algorithm to obtain U

      DO 70 K=1,N
      DO 70 I=1,NMIN1
      IF(I.EQ.K) GO TO 70
      DIK=U(I,K)
      DO 70 J=I+1,N
```

```
          IF(J.EQ.K) GO TO 70
          ISUM=DIK+U(K,J)
          IF(ISUM.GE.U(I,J)) GO TO 70
          U(I,J)=ISUM
          U(J,I)=ISUM
          BETA(I,J)=BETA(K,J)
          BETA(J,I)=BETA(K,I)
   70     CONTINUE
**
**        Find the ordering of arcs
**
          DO 90 I=1,N
          DO 90 J=1,N
          IF(I.EQ.J) GO TO 90
          K=J
   80     L=PRED(I,K)
          LL=COUNT(K,L)
          M(I,J)=M(LL)+1
          K=L
          IF(K.NE.I) GO TO 80
   90     CONTINUE
          DO 110 I=1,M
          MIN=INFTY
          DO 100 J=1,M
          IF(MIN.LE.MU(J)) GO TO 100
          MIN=MU(J)
          K=J
  100     CONTINUE
          IORDER(I)=K
          MU(K)=INFTY
  110     CONTINUE
          X=SINK
          LINK(X)=INFTY
          NODES=0
          NODESF=0
          NFS=0
          NIMP=0
          TOP=0
          MDEPTH=0
          INC=0
**
**        Main loop
**
  120     NODES=NODES+1
          CALL READ(NT,2)
          IF(I.NE.GO.JU.NIT) GO TO 690
          LBP=0
          DO 130  I=1,NIT1
          DO 130  J=I+1,N
          DET=MIN(AT(U(I,J),UO(I,J))
          LB =(GRU+D)J*R(I,J)
          DEJ=DEJ-R(I,J)
          ASU(I,J)=DET
          RSU(I,J)=DEJ
          LB =(LB +EPSIL)*LBO
          IF(RUB.GE.ZSTAR) GO TO 520
**
**        Lower Bound calculation
**
**        Determine MU vector.
**
          LAST=0
          TOTNGT=0
          MCOST=-INFTY
          KINIT=INC
          DO 210 LZ=1,M
          S=IORDER(LZ)
          IF(SINT.IS(S).NE.0) GO TO 210
          S=INODE(S)
          D=JNODE(S)
```

```
      DPQ=Q(P,Q)
      IF(Q(P,Q).LT.DPQ) GO TO 210
      KOUNT=KOUNT+1
      Q(P,Q)=INFTY
      Q(Q,P)=INFTY
      V1=0
      V2=N+1
      DO 150 I=1,N
      IF((Q(I,P)+DPQ).NE.U(I,Q)) GO TO 140
      V1=V1+1
      SETV(V1)=I
      GO TO 150
140   IF((DPQ+U(Q,I)).NE.U(P,I)) GO TO 150
      V2=V2-1
      SETR(V2)=I
150   CONTINUE
      NPAIRS=0
      WEIGHT=0
      ICOST=0
      DO 160 I1=1,N1
      I=SETV(I1)
      DO 160 I2=N2,N
      L=SETR(I2)
      ISUM=Q(I,Q)+DPQ+U(Q,L)
      IF(ISUM.NE.U(I,L)) GO TO 160
      NPAIRS=NPAIRS+1
      INODE(NPAIRS)=I
      LNODE(NPAIRS)=L
      V(NPAIRS)=ISUM
      Q(I,L)=U(I,L)
      U(L,I)=U(L,I)
160   CONTINUE
      Q(P,Q)=DPQ
      Q(Q,P)=DPQ
      IF(NPAIRS.EQ.0) GO TO 210
      DO 170 K=1,N
      DO 170 J=1,NPAIRS
      I=INODE(J)
      L=LNODE(J)
      ISUM=U(I,K)+U(K,L)
      IF(ISUM.GE.U(I,L)) GO TO 170
      U(I,L)=ISUM
      U(L,I)=ISUM
170   CONTINUE
      DO 180 J=1,NPAIRS
      I=INODE(J)
      L=LNODE(J)
      KK=V(J)
      ISUM=U(I,L)-KK
      IF((LINK(I).EQ.0).OR.(LINK(L).EQ.0)) ICOST=ICOST+ISUM*R(I,L)
      U(I,L)=KK
      U(L,I)=KK
      KK=ISUM-BETA(I,L)
      IF(KK.LE.0) GO TO 180
      BETA(I,L)=ISUM
      BETA(L,I)=ISUM
      WEIGHT=WEIGHT+KK*R(I,L)
180   CONTINUE
      IF(MCOST.GE.ICOST) GO TO 200
      IF(LINK(P).EQ.0) GO TO 190
      IF(LINK(Q).NE.0) GO TO 200
      MCOST=ICOST
      PP=P
      QQ=Q
      GO TO 200
190   IF(LINK(Q).EQ.0) GO TO 200
      MCOST=ICOST
      PP=Q
      QQ=P
200   IF(WEIGHT.EQ.0) GO TO 210
```

```
        LAST=LAST+1
        MU(LAST)=WEIGHT
        ARC(LAST)=S
        TOTWGT=TOTWGT+WEIGHT
210     CONTINUE
        IF(KOUNT.GT.NMIN1) GO TO 230
**
**      The node is fathomed
**
        NODESF=NODESF+1
        Z=LBD
        I=IVT
        DO 220 IZ=1,M
        IF(SEPLST(IZ).NE.0) GO TO 220
        P=INDX(IZ)
        Q=HINDX(IZ)
        IF(J(P,Q).GT.D(P,J)) GO TO 220
        I=I+1
        HD(I)=P
        TL(I)=Q
220     CONTINUE
        GO TO 480
230     CONTINUE
**
**      Form the Maximum Spanning Tree
**
        IF((INC+LAST).LE.NMIN1) GO TO 340
        MU(LAST+1)=-INFTY
        FIRST=LAST/2
        KOUNT=INC
        NTGT=0
        DO 240 I=1,N
240     FATHER(I)=PARENT(I)
250     J=FIRST
        TEMP=MU(J)
        ITEMP=ARC(J)
260     K=J+J
        IF(K.GT.LAST) GO TO 270
        IF(MU(K).GT.MU(K+1)) K=K+1
        IF(MU(K).LE.TEMP) GO TO 270
        MU(J)=MU(K)
        ARC(J)=ARC(K)
        J=K
        GO TO 260
270     MU(J)=TEMP
        ARC(J)=ITEMP
        FIRST=FIRST-1
        IF(FIRST.GT.0) GO TO 250
        K=ARC(1)
        I=HNODE(K)
        K=TNODE(K)
280     IF(FATHER(I).LT.0) GO TO 290
        I=FATHER(I)
        GO TO 280
290     IF(FATHER(K).LT.0) GO TO 300
        K=FATHER(K)
        GO TO 290
300     IF(I.EQ.K) GO TO 330
        KK=FATHER(I)+FATHER(K)
        IF(FATHER(I).GT.FATHER(K)) GO TO 310
        FATHER(K)=I
        FATHER(I)=KK
        GO TO 320
310     FATHER(I)=K
        FATHER(K)=KK
320     KOUNT=KOUNT+1
        TOTWGT=TOTWGT+MU(1)
330     MU(1)=MU(LAST)
        ARC(1)=ARC(LAST)
        LAST=LAST-1
```

```
            IF(NCOUNT.LE.NMIN)) GO TO 250
      F=I  +TOPWGT-TWEIGT
      RLB=(1.0+EPSLON)*LB
      IF(RLB.GE.ZSTAR) GO TO 520
340   CONTINUE
**
**      Form the SEPP
**
      P=INFTY
      Q=Q0
      I=K
      DPQ=D(PP,QQ)
350   IF(I.EQ.INFTY) GO TO 370
      IF(D(I,Q).EQ.INFTY) GO TO 360
      IF(I.EQ.PP) GO TO 360
      SEPP(I)=P
      P=I
360   JP(I,Q)=JP(I,PP)+DPQ
      JP(Q,I)=UP(I,Q)
      I=LINK(I)
      GO TO 350
370   CONTINUE
**
**      Delete all arcs (P,Q) , P in SEPP
**
      IPAIRS=0
380   IF(P.EQ.INFTY) GO TO 430
      DPQ=D(P,Q)
      D(P,Q)=INFTY
      D(Q,P)=INFTY
      N1=0
      N2=N+1
      DO 400 I=1,N
      IF((J(I,P)+DPQ).NE.J(I,Q)) GO TO 390
      N1=N1+1
      SEPV(N1)=I
      GO TO 400
390   IF((DPQ+U(Q,I)).NE.U(P,I)) GO TO 400
      N2=N2-1
      SEPV(N2)=I
400   CONTINUE
      IF(N1.LT.1) GO TO 420
      IF(N2.GT.N) GO TO 420
      DO 410 I1=1,N1
      I=SEPV(I1)
      DO 410 I2=N2,N
      L=SEPV(I2)
      IF(RMATR(I,L).GE.NODES) GO TO 410
      IF((J(I,Q)+DPQ+J(Q,L)).NE.U(I,L)) GO TO 410
      IPAIRS=IPAIRS+1
      IPQRSC(IPAIRS)=I
      LPQRSC(IPAIRS)=L
      RMATR(I,L)=NODES
      RMATR(L,I)=NODES
410   CONTINUE
420   P=SEPP(P)
      GO TO 380
430   IF(IPAIRS.EQ.0) GO TO 460
      DO 440 J=1,IPAIRS
      I=IPQRSC(J)
      L=LPQRSC(J)
      J(I,L)=D(I,L)
      J(L,I)=D(L,I)
440   DO 450 K=1,N
      DO 450 J=1,IPAIRS
      I=IPQRSC(J)
      L=LPQRSC(J)
450   J=U(L,K)+J(K,L)
```

```
         IF(ISUM.GE.U(I,L)) GJ TO 450
         J(I,L)=ISUM
         J(L,I)=ISUM
  450    CONTINUE
  460    CONTINUE
         PARENT(JJ)=SEED
         INC=INC+1
         PARENT(SEED)=-INC-1
         TD(INC)=JJ
         TL(INC)=PS
         K=ARCNUM(PP,JJ)
         STATUS(K)=1
         TOP=TOP+1
         IF(TOP.GT.MDEPTH) MDEPTH=TOP
         STACK(TOP,1)=PS
         STACK(TOP,2)=JJ
         LINK(JJ)=K
         X=JJ
         IF(INC.LT.NMIN1) GJ TO 120
**
**       A feasible solution is obtained
**       Update incumbent if necessary
**
         NFS=NFS+1
         Z=0
         DO 470 I=1,NMIN1
         DO 470 J=I+1,N
  470    Z=Z+R(I,J)*U(I,J)
  480    IF(Z.GE.ZSTAR) GD TO 510
         ITMP=ITMP+1
         WRITE(33,490) ITMP,Z,ZSTAR,NFS
  490    FORMAT(3X,' ITMP Z ZSTAR NFS ',4I12)
         ZS=Z
         DO 500 I=1,NMIN1
         ITAD(I)=TD(I)
         ITAL(I)=TL(I)
  500    CONTINUE
  510    CONTINUE
**
**       Backtrack
**
  520    IF(TOP.EQ.0) GO TO 690
         PS=STACK(TOP,1)
         JJ=STACK(TOP,2)
         K=LINK(JJ)
         IF(STATUS(K).EQ.2) GO TO 640
         STATUS(K)=2
         Q=JJ
         P=X
  530    IF(P.EQ.INFTY) GO TO 580
         TD(P,0)=0
         TD(0,P)=0
         K=ARCNUM(P,0)
         IF(K.EQ.0) GO TO 570
         IF(STATUS(K).EQ.2) GO TO 570
         QP=PARENT(K)
         O(P,0)=QP
         O(0,P)=QP
         N1=0
         N2=N+1
         DO 550 I=1,N
         KK=J(I,P)+DPQ
         IF(KK.GE.J(I,0)) GO TO 540
         J(I,0)=KK
         J(0,I)=KK
         N1=N1+1
         ISTACK(N1)=I
  540    KK=DPQ+J(0,I)
         IF(KK.GE.J(P,0)) GO TO 550
         J(P,I)=KK
```

```
        J(1,P)=KK
        N2=N2-1
        SETN(N2)=I
550     CONTINUE
        IF(N1.EQ.0) GO TO 570
        IF(N2.GT.N) GO TO 570
        DO 560 I1=1,N1
        I=SETN(I1)
        DO 560 I2=N2,N
        L=SETN(I2)
        KK=J(I,P)+DPQ+U(P,L)
        IF(KK.GE.U(I,L)) GO TO 560
        J(I,L)=KK
        J(L,I)=KK
560     CONTINUE
570     P=LINK(P)
        GO TO 530
580     CONTINUE
**
**      Delete arc (PP,QQ)
**
        DPQ=D(PP,QQ)
        D(PP,QQ)=INFTY
        D(QQ,PP)=INFTY
        NPAIRS=0
        N1=0
        N2=N+1
        DO 600 I=1,N
        IF((J(I,PP)+DPQ).NE.U(I,QQ)) GO TO 590
        N1=N1+1
        SETN(N1)=I
        GO TO 600
590     IF((DPQ+J(QQ,I)).NE.U(PP,I)) GO TO 600
        N2=N2-1
        SETN(N2)=I
600     CONTINUE
        IF(N1.EQ.0) GO TO 630
        IF(N2.GT.N) GO TO 630
        DO 610 I1=1,N1
        I=SETN(I1)
        DO 610 I2=N2,N
        L=SETN(I2)
        IF((J(I,PP)+DPQ+U(QQ,L)).NE.U(I,L)) GO TO 610
        NPAIRS=NPAIRS+1
        INODE(NPAIRS)=I
        JNODE(NPAIRS)=L
        J(I,L)=D(I,L)
        J(L,I)=D(L,I)
610     CONTINUE
        IF(NPAIRS.EQ.0) GO TO 630
        DO 620 K=1,N
        DO 620 J=1,NPAIRS
        I=INODE(J)
        L=JNODE(J)
        ISUM=J(I,K)+U(K,L)
        IF(ISUM.GE.U(I,L)) GO TO 620
        J(I,L)=ISUM
        J(L,I)=ISUM
620     CONTINUE
630     SPARSE(D(I,J))=-1
        INC=INC-1
        PAREN(SETD)=-INC-1
        K=LINK(QQ)
        LINK(QQ)=I
        GO TO 120
640     K=MINIM(PP,QQ)
        SPARSE(K)=0
        DPQ=LENGTH(K)
**
```

```
          D(PP,QQ)=DPQ
          D(QQ,PP)=DPQ
          IF(J(PP,QQ).LT.DPQ) GO TO 680
          N1=0
          N2=I+1
          DO 650 I=1,N
          ISUM=J(I,PP)+DPQ
          IF(J(I,QQ).LE.ISUM) GO TO 650
          J(I,QQ)=ISUM
          J(QQ,I)=ISUM
          N1=N1+1
          SETN(N1)=I
          GO TO 660
650       ISUM=DPQ+J(QQ,I)
          IF(J(PP,I).LE.ISUM) GO TO 660
          J(PP,I)=ISUM
          J(I,PP)=ISUM
          N2=N2-1
          SETN(N2)=I
660       CONTINUE
          IF(N1.EQ.0) GO TO 680
          IF(N2.GT.N) GO TO 680
          DO 670 I1=1,N1
          I=SETN(I1)
          DO 670 I2=N2,N
          L=SETN(I2)
          ISUM=J(I,PP)+U(PP,L)
          IF(ISUM.GE.J(I,L)) GO TO 670
          J(I,L)=ISUM
          J(L,I)=ISUM
670       CONTINUE
680       TOP=TOP-1
          GO TO 520
690       CONTINUE
          L   PII  II(IIMP2)
          II  I    IN+TIM2-TIME1
          II  II(3)/100)
700       FORMAT(1X/' Branch and Bound Algorithm ')
          FORMAT(3X,I10,  N,I5.0),IGN,EPSLON,SEED,ZSTAR,ITIVE,
         1        NODES,FCMSF,FS,NIMP,DEPTH
710       FORMAT(3X,' No. of Nodes ',6X,' = ',I6,' No. of Arcs     = ',I6/3X,
         1      '  Seed                  = ',I6,' NG Number       = ',I6/3X,
         2      '  Epsilon',14X,'= ',F6.3,' Seed Node       = ',I6/3X,
         3      '  Zstar                 = ',I12,' CPU Time In Msec = ',I8/3X,
         4      '  Nodes Examined        = ',I6,' Nodes Fathomed  = ',I6/3X,
         5      '  No. of Feasible Solutions = ',I6/3X,
         6      '  No. of Improvements = ',I6,' Maximum Depth    = ',I6)
          WRITE(32,720) (IER(I),IAII(I),I=1,NMIN1)
720       FORMAT(3X,' Solution - ',I4/10(15X,20I4))
          IF(.GT.0) WRITE(32,730)
730       FORMAT(/3X,' Time Limit Exceeded ')
          STOP
          END
**
```

```
**
**      Program for TREE - BUILDING Algorithm
**
**                    V.V.S.Murty.
**
**      Input Data       :
**
**      V        :       Number of nodes
**      A        :       Number of arcs
**      ISEED    :       Seed value
**      NGN      :       1      for Euclidean Networks
**                       2      for Random Networks
**
**      Important Variables    :
**
**      SEED     :       Starting seed node
**      INFTY    :       A large number : 1000000
**      D(V,V)   :       Distance matrix
**      R(V,V)   :       Requirement matrix
**      T(V,V)   :       Shortest path distance matrix
**      X        :       Starting node of set S
**      Y        :       Starting node of set SBAR
**      LINK(T)  :       Gives the next node in the set
**                       INFTY if that is the last node
**      V(I)     :       Communication of a node with
**                       nodes in the other set
**      NTOTAL   :       Total communication over cutset (S,SBAR)
**      LABEL(V) :       Label given to a node
**      Z        :       Objective function value
**      HEAD(V)  :       Contains the head node of
**                       arcs of the spanning tree
**      TAIL(V)  :       Contains the tail node of
**                       arcs of the spanning tree
**      R1,R2    :       Range of R
**      X1,X2    :       Plane size for X coordinate
**      Y1,Y2    :       Plane size for Y coordinate
**      NODDEG   :       Minimum degree of node
**                       that is required
**      ORDER(4) :       Contains 4 nodes supplied
**                       Network Generator
**
        INTEGER DPO,P,Q,RR,SEED,X,Y,NTOTAL,
       1       TIME1,NFTY,Z,ORDER(4),
       2       LINK(50),LABEL(50),HEAD(50),TAIL(50),
       3       T(50),D(50,50),R(50,50),D(50,50),
       4       R1,R2,X1,X2,Y1,Y2
        DATA INFTY,ZSTAR/1000000,1000000000/
        DATA NODDEG,R1,R2,X1,X2,Y1,Y2/2,1,100,0,200,0,200/
        READ(21,*) V,M,ISEED,NGN
        GO TO (10,20),NGN
10      CALL NG1(V,M,ISEED,NODDEG,R1,R2,X1,X2,Y1,Y2,D,R,ORDER)
        GO TO 30
20      CALL NG2(V,M,ISEED,NODDEG,R1,R2,X1,X2,Y1,Y2,D,R,ORDER)
30      WRITE(22,40)
40      FORMAT(3X,' Two Phase Heuristic Algorithm')
        WRITE(22,50) V,M,ISEED,NGN
50      FORMAT(3X,' No.of nodes = ',I5,' No.of arcs = ',I5/3X,
       1         Seed value = ',I5,' Network Generator Number = ',I4/)
        WRITE(22,60)
60      FORMAT(3X,' Phase I is followed by Phase II '/)

        NMINI=V-1
        DO 350 IZ=1,4
        CALL RTIME(TIME1)

**
**      Initialization
**
        SEED=ORDER(IZ)
        X=SEED
```

```
          LINK(K)=INFTY
          I=MOD(X,N)+1
          Y=I
   70     I=MOD(I,N)+1
          IF(J.EQ.K) GO TO 80
          LINK(I)=I
          I=I
          GO TO 70
   80     LINK(I)=INFTY
          DO 90 I=1,N
          W(I)=0
          DO 90 I=1,N
   90     J(I,J)=D(I,J)
          WTOTAL=0
**
**        Apply FLOYD's Algorithm to obtain U
**
          DO 100 K=1,N
          DO 100 I=1,NMIN1
          IF(I.EQ.K) GO TO 100
          KK=J(I,K)
          DO 100 J=I+1,N
          IF(J.EQ.K) GO TO 100
          ISUM=KK+J(K,J)
          IF(ISUM.GE.J(I,J)) GO TO 100
          J(I,J)=ISUM
          J(J,I)=ISUM
  100     CONTINUE
**
**        Add n-1 arcs one by one
**        to form a spanning tree
**
          DO 280 KOUNT=1,NMIN1
          J=Y
          ISUM=0
  110     IF(J.EQ.INFTY) GO TO 120
          RR=R(X,J)
          W(J)=W(J)+RR
          ISUM=ISUM+RR
          J=LINK(J)
          GO TO 110
  120     W(X)=ISUM
          WTOTAL=WTOTAL+ISUM
**
**        Select the branching arc (P,Q)
**        Form the labels of nodes
**
          I=X
  130     IF(I.EQ.INFTY) GO TO 160
          J=X
          ISUM=0
  140     IF(J.EQ.INFTY) GO TO 150
          ISUM=ISUM+W(J)*J(I,J)
          J=LINK(J)
          GO TO 140
  150     LABEL(I)=ISUM
          I=LINK(I)
          GO TO 130
  160     CONTINUE
          I=Y
  170     IF(I.EQ.INFTY) GO TO 200
          J=Y
          ISUM=0
  180     IF(J.EQ.INFTY) GO TO 190
          ISUM=ISUM+W(J)*J(I,J)
          J=LINK(J)
          GO TO 180
  190     LABEL(I)=ISUM
          I=LINK(I)
          GO TO 170
```

```
  200      CONTINUE
**
**       Select the arc (P,Q)
**
         MIN=1000000
         K=INFTY
         I=Y
  210    IF(J.EQ.INFTY) GO TO 250
         I=X
  220    IF(I.EQ.INFTY) GO TO 240
         IF(U(I,J).LT.D(I,J)) GO TO 230
         ISUM=LABEL(I)+WTOTAL*D(I,J)+LABEL(J)
         IF(ISUM.GE.MIN) GO TO 230
         MIN=ISUM
         P=I
         Q=J
         KK=K
  230    I=LINK(I)
         GO TO 220
  240    K=J
         J=LINK(J)
         GO TO 210
  250    CONTINUE
**
**       Update N,U and WTOTAL
**
         DPQ=D(P,Q)
         I=X
  260    IF(I.EQ.INFTY) GO TO 270
         RR=R(I,Q)
         N(I)=N(I)-RR
         WTOTAL=WTOTAL-RR
         U(I,Q)=U(I,P)+DPQ
         U(Q,I)=U(I,Q)
         I=LINK(I)
         GO TO 260
  270    CONTINUE
         TAIL(KOUNT)=P
         HEAD(KOUNT)=Q
         IF(KK.EQ.INFTY) Y=LINK(Y)
         IF(KK.NE.INFTY) LINK(KK)=LINK(Q)
         LINK(Q)=X
         X=Q
  280    CONTINUE
         Z=0
         DO 290 I=1,NMIN1
         DO 290 J=I+1,N
  290    Z=Z+R(I,J)*U(I,J)
         CALL RTIME(TIME2)
         TIME2=TIME2-TIME1
         WRITE(22,300) SEED,Z,TIME2
  300    FORMAT(3X,' Seed node = ',I5,' Obj.fn.value = ',I10,
        1          ' Time in Msec = ',I6)
  310    CONTINUE
         CALL IMPROV(N,ITER,ITIME,Z,HEAD,TAIL,D,R,U)
         WRITE(22,320) Z,ITIME,ITER
  320    FORMAT(3X,' Obj.fn.value = ',I10,' Time in Msec = ',I6,
        1          ' No.of iterations = ',I5/)
         WRITE(23,330) SEED
  330    FORMAT(3X,' Seed node = ',I5)
         WRITE(23,340) (TAIL(I),HEAD(I),I=1,NMIN1)
  340    FORMAT(3X,' Solution ',(50I4))
  350    CONTINUE
         STOP
         END
**
**       Program for TREE - IMPROVEMENT Algorithm
**
         SUBROUTINE IMPROV(N,ITER,ITIME,Z,HEAD,TAIL,D,R,U)
         INTEGER N,ITER,ITIME,Z,HEAD(50),TAIL(50),
```

```
     1          D(50,50),R(50,50),U(50,50),
     2          TJJVT,DIJ,DP),P,Q,PP,QQ,RR,
     3          PCOST,TIME1,TIME2,WTOTAL
     4          LABEL(50),SETN(50),W(50),
       IMEN1=N-1
       INFTY=1000000
       CALL RTIME(TIME1)
       ITER=0
       K=0
**  10 ITER=ITER+1

       DO 100 COUNT=1,NMIN1
       K=MOD(K,NMIN1)+1
       P=TAIL(K)
       Q=HEAD(K)
**
**     (P,Q) is the tree arc considered
**     for exchange with a non tree arc
**
       DPQ=D(P,Q)
       N1=0
       N2=N+1
       DO 3) I=1,N
       W(I)=0
       IF((U(I,P)+DPQ).NE.U(I,Q)) GO TO 20
       N1=N1+1
       SETN(N1)=I
       GO TO 30
   20  N2=N2-1
       SETN(N2)=I
   30  CONTINUE
       WTOTAL=0
**
**     Find the non tree arc (PP,QQ)
**
       DO 40 I1=1,N1
       I=SETN(I1)
       DO 40 I2=N2,N
       J=SETN(I2)
       RR=R(I,J)
       W(I)=W(I)+RR
       W(J)=W(J)+RR
   40  WTOTAL=WTOTAL+RR
       DO 60 I1=1,N1
       I=SETN(I1)
       ISUM=0
       DO 50 I2=1,N1
       J=SETN(I2)
   50  ISUM=ISUM+W(J)*U(I,J)
   60  LABEL(I)=ISUM
       DO 80 I1=N2,N
       I=SETN(I1)
       ISUM=0
       DO 70 I2=N2,N
       J=SETN(I2)
   70  ISUM=ISUM+W(J)*U(I,J)
   80  LABEL(I)=ISUM
       PCOST=LABEL(P)+WTOTAL*DPQ+LABEL(Q)
       MCOST=PCOST
       DO 90 I1=1,N1
       I=SETN(I1)
       DO 90 I2=N2,N
       J=SETN(I2)
       KK=D(I,J)
       IF(KK.EQ.INFTY) GO TO 90
       ISUM=LABEL(I)+WTOTAL*KK+LABEL(J)
       IF(ISUM.GE.MCOST) GO TO 90
       MCOST=ISUM
       PP=I
       QQ=J
```

```
  90      CONTINUE
         IC)ST=PCOST-NCOST
         IF(NCOST.GE.J) GO TO 110
 100     CONTINUE
         GO TO 130
 110     CONTINUE
**
**       Exchange (P,Q) with (PP,QQ)
**       and update Z,HEAD,TAIL and U
**
         Z=Z-NCOST
         TAIL(K)=PP
         HEAD(K)=QQ
         OP)=)(PP,QQ)
         DO 120 I1=1,N1
         I=SORT1(I1)
         DO 120 I2=N2,N
         I2=   (I2)
         U(I,J)=U(I,PP)+OPQ+U(QQ,J)
 120     U(J,I)=U(I,I)
 130     CONTINUE RETAG(TIMG2)
         IF(NG=TIME)-TIME)
         RETURN
**       END
```

```
**
**        Subroutines for Euclidean and Random
**        Network Generators NG1 and NG2
**
**        Euclidean Network Generator
**
**        SUBROUTINE NG1(N,M,ISEED,NODDEG,R1,R2,X1,X2,Y1,Y2,A,R,ORDER)

**        INTEGER R1,R2,X1,X2,Y1,Y2,RSUM,ORDER(4),RMAX(4),
          1         DEGREE(50),QUAD(50),TOTAL(50),X(50),Y(50),
          2         A(50,50),R(50,50)
**
          INFTY=1000000
          N1=N-1
          MX=(X1+X2)/2
          MY=(Y1+Y2)/2
          IF(2*M.LE.N*N1) GO TO 20
          WRITE(22,10)
   10     FORMAT(3X,' M > N(N-1)/2 ')
          STOP
**
   20     CONTINUE
          CALL SETRAN(ISEED)
          DO 40 I=1,N1
          DO 30 J=I+1,N
          A(I,J)=INFTY
          A(J,I)=INFTY
          KK=IRAN(R1,R2)
          R(I,J)=KK
   30     R(J,I)=KK
          A(I,I)=0
   40     R(I,I)=0
          A(N,N)=0
          R(N,N)=0
**
          DO 60 I=1,N
          DEGREE(I)=0
          RSUM=0
          DO 50 J=1,N
   50     RSUM=RSUM+R(I,J)
   60     TOTAL(I)=RSUM
**
          DO 100 I=1,N
   70     IX=IRAN(X1,X2)
          IY=IRAN(Y1,Y2)
          IF(I.EQ.1) GO TO 90
          DO 80 J=1,I-1
   80     IF((IX.EQ.X(J)).AND.(IY.EQ.Y(J))) GO TO 70
   90     CONTINUE
          X(I)=IX
          Y(I)=IY
          KK=1
          IF(IX.GE.MX) KK=3
          IF(IY.GE.MY) KK=KK+1
          QUAD(I)=KK
  100     CONTINUE
**
          DO 120 I=1,M
  110     J=IRAN(1,N)
          K=IRAN(1,N)
          IF(A(J,K).NE.INFTY) GO TO 110
          DEGREE(J)=DEGREE(J)+1
          DEGREE(K)=DEGREE(K)+1
          IX=X(J)-X(K)
          IY=Y(J)-Y(K)
          XX=IX*IX+IY*IY
          KK=IFIX(SQRT(XX))
          A(J,K)=KK
          A(K,J)=KK
  120     CONTINUE
```

```
**
        DO 150 I=1,N
  130    IF(DEGREE(I).GE.NODDEG) GO TO 150
  140    J=IRAN(I,J)
         IF(A(I,J).NE.INFTY) GO TO 140
         DEGREE(I)=DEGREE(I)+1
         DEGREE(J)=DEGREE(J)+1
         M=M+1
         IX=X(I)-X(J)
         IY=Y(I)-Y(J)
         XX=IX*IX+IY*IY
         KK=IFIX(SQRT(XX))
         A(I,J)=KK
         A(J,I)=KK
         GO TO 130
**150    CONTINUE

         DO 190 I=1,N
         MAX=0
         DO 160 J=1,N
         IF(ORDER(J).NE.1) GO TO 160
         IF(MAX.GE.TOTAL(J)) GO TO 160
         MAX=TOTAL(J)
         KK=J
  160    CONTINUE
         IF(MAX.NE.0) GO TO 180
         DO 170 J=1,N
         IF(MAX.GE.TOTAL(J)) GO TO 170
         MAX=TOTAL(J)
         KK=J
  170    CONTINUE
  180    CONTINUE
         RMAX(I)=MAX
         ORDER(I)=KK
         TOTAL(KK)=0
**190    CONTINUE

         DO 210 I=1,3
         MAX=0
         DO 200 J=1,4
         IF(MAX.GE.RMAX(J)) GO TO 200
         MAX=RMAX(J)
         K=J
  200    CONTINUE
         KK=ORDER(I)
         ORDER(I)=ORDER(K)
         ORDER(K)=KK
         RMAX(K)=RMAX(I)
         RMAX(I)=0
  210    CONTINUE
         RETURN
         END
**
**       Random Network Generator
**
**       SUBROUTINE NG2(N,M,ISEED,NODDEG,R1,R2,A1,A2,Y1,Y2,A,R,ORDER)
         INTEGER A1,A2,R1,R2,RSUM,Y1,Y2,ORDER(4),RMAX(4),
        2        DEGREE(50),TOTAL(50),A(50,50),R(50,50)
**
         INFTY=1000000
         N1=N-1
         IF(2*M.LE.N*N1) GO TO 20
         WRITE(22,10)
   10    FORMAT(3X,' M > N(N-1)/2 ')
         STOP
**
   20    CONTINUE
         CALL SETRAN(ISEED)
         DO 40 I=1,N1
```

```
        DO 30 J=I+1,N
        A(I,J)=INFTY
        A(J,I)=INFTY
        KK=IRAN(R1,R2)
        R(I,J)=KK
   30   R(J,I)=KK
        A(I,I)=0
   40   R(I,I)=0
        A(N,N)=0
        R(N,N)=0
**
        DO 60 I=1,N
        DEGREE(I)=0
        RSUM=0
        DO 50 J=1,N
   50   RSUM=RSUM+R(I,J)
   60   TOTAL(I)=RSUM
**
        DO 80 I=1,M
   70   I=IRAN(1,N)
        K=IRAN(1,N)
        IF(A(I,K).NE.INFTY) GO TO 70
        DEGREE(J)=DEGREE(J)+1
        DEGREE(K)=DEGREE(K)+1
        KK=IRAN(A1,A2)
        A(I,K)=KK
        A(K,I)=KK
   80   CONTINUE
**
        DO 110 I=1,N
   90   IF(DEGREE(I).GE.NODDEG) GO TO 110
  100   J=IRAN(1,N)
        IF(A(I,J).NE.INFTY) GO TO 100
        DEGREE(I)=DEGREE(I)+1
        DEGREE(J)=DEGREE(J)+1
        M=M+1
        KK=IRAN(A1,A2)
        A(I,J)=KK
        A(J,I)=KK
        GO TO 90
**110   CONTINUE
**
        DO 130 I=1,N
        MAX=0
        DO 120 J=1,N
        IF(MAX.GT.TOTAL(J)) GO TO 120
        MAX=TOTAL(J)
        K=J
  120   CONTINUE
        ORDER(I)=K
  130   TOTAL(K)=0
        RETURN
        END
**
**
**      Function to Generate a Uniform Random
**      Numbers in the range IA and IB
**
        INTEGER FUNCTION IRAN(IA,IB)
        K=IA+(IB-IA+1)*RAN(X)
        IF(K.GT.IB)K=IB
        IRAN=K
        RETURN
**      END
```

-EXA